MASTER EN SCIENCES INFORMATIQUES

Animation graphique de tâches de bureau

Simoens, Marc; Van Pevenaeyge, Olivier

Award date:
1987

Awarding institution:
Universite de Namur

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Animation graphique de tâches de bureau

Mémoire présenté par
Marc Simoens et Olivier Van Pevenaeyge
en vue de l'obtention du titre de
Licencié et Maître en Informatique.

Promoteur : Mr. Roland Lesuisse.

Année académique 1986 - 1987.

Erratum

Ligne 7 et 11 : lire "également" au lieu de "égalemment"

Remerciements

Qu'il nous soit ici permis de remercier plus que vivement Monsieur Roland Lesuisse, promoteur de ce mémoire, qui par ses nombreux conseils et attentives lectures nous a permis d'améliorer grandement la qualité de notre rédaction et qui, par ses encouragements, sut entretenir notre motivation durant cette longue année.

Nous remercions également Nadine Dachouffe pour ses judicieuses remarques sur une version préliminaire de ce mémoire ainsi que pour avoir, par ses travaux antérieurs, jeté les bases de celui-ci.

Merci également à Monsieur Jean-Luc Hainaut pour ses précisions concernant la notion de sous-typage.

Nous n'oublions pas les personnes contactées lors de notre recherche de documentation ainsi que celles qui, de près ou de loin, ont collaboré au présent travail.

Nous tenons finalement à remercier nos parents respectifs sans l'aide et le soutien desquels, l'accomplissement de nos études n'aurait pas été possible.

TABLE DES MATIERES

Chapitre 1
Introduction

CHAPITRE 1 : INTRODUCTION

Ce travail fait partie d'un vaste projet dont le contexte général est la bureautique. Dans une première section, nous ferons donc une brève description de ce qu'est la bureautique et dans une deuxième section, nous décrirons les objectifs de notre mémoire ainsi que le projet dans lequel il s'insère.

1.1 Description générale de la bureautique [BLAS 82], [HINE 85]

La grande innovation de ces dernières années est l'introduction de l'informatique et de ses technologies dans l'environnement du bureau. L'informatique ne s'intéresse plus seulement aux activités traditionnelles d'automatisation de procédures administratives classiques (paie, comptabilité, facturation,...) mais également aux activités individuelles des bureaux eux-mêmes (préparation de documents, communication, gestion de dossiers, agencement de réunions,...).

C'est ainsi que le concept de bureautique a fait son apparition aux Etats-Unis au début des années 70 avec l'expression "office automation". L'équivalent français serait "automatisation du bureau" ou "automatisation du tertiaire". Ce n'est qu'en 1977 que le terme bureautique fait son apparition et c'est en 1978 qu'il sera officialisé.

Le terme de bureautique est un concept d'origine récente et se révèle par conséquent difficile à définir. Beaucoup d'auteurs s'y sont essayés mais en général, ils tombent tous dans le même travers. Ils considèrent que la bureautique a pour seul objectif de faire exécuter en tout ou en partie des tâches de bureau par des machines. Ils oublient complètement la composante humaine au profit des techniques et des procédés.

La définition que nous allons donner a été proposée par des auteurs qui croient que la bureautique doit s'appliquer à toutes les activités de bureau, et qu'elle est autant concernée par les conséquences humaines et sociales qu'elle entraîne que par les outils et procédures qu'elle met en jeu.

> "La bureautique désigne l'assistance aux travaux de bureau, procurée par des moyens et des procédures faisant appel aux techniques de l'informatique, des télécommunications et de l'organisation administrative et, de façon générale, à tout ce qui concourt à la logistique du bureau et de son environnement. Plus conceptuellement, la bureautique intéresse le système d'information individuel de toute personne travaillant dans un bureau, sans exiger d'elle d'autres connaissances que celles de son savoir-faire professionnel" [BLAS 82].

Les principaux domaines d'application de la bureautique se situent donc dans les bureaux; il est dès lors intéressant et nécessaire de définir le bureau.

"Un bureau typique est une cellule de l'organisation ayant une activité commune s'apparentant à un service. Sa dimension est réduite dans l'espace (quelques pièces) et en nombre de personnes. Un même bureau peut comprendre diverses catégories de personnel : cadres, techniciens, agents, employés, secrétaires, dactylos. Leurs postes de travail sont en général voisins. C'est son activité qui fait l'unité du bureau même si les tâches accomplies par les divers acteurs sont de nature différentes. Un bureau dispose d'une relative autonomie pour l'accomplissement de ses activités, qu'elles soient de production, de transformation, de création ou autres. Autrement dit un bureau est une cellule suffisament autonome pour prendre lui-même en charge la responsabilité de l'accomplissement de ses activités : ajustement de son rythme de travail à la charge, définition de la qualité des travaux, etc ..." [BLAS 82].

Chaque personne dans un bureau travaille, avons-nous dit, sur son système d'information. Il nous reste à définir ce que nous désignons sous ce terme. Par système d'information d'une organisation, nous entendrons une construction formée d'ensembles :

- d'informations, représentations partielles de faits qui intéressent l'organisation,
- de traitements, procédés d'acquisition, de mémorisation, de recherche, de communication et de transformation des informations et
- de ressources, humaines, techniques et organisationnelles qui en assurent le fonctionnement. [BOPI 83]

Chaque personne travaille sur un système d'information car une des premières fonctions d'un bureau est de garder trace de ce qui se passe : savoir ce qui est arrivé dans le passé, et bien entendu pouvoir le retrouver en temps utile. Stocker, faire circuler et restituer l'information sont devenus les principaux objectifs des bureaux, dont on attend effectivement qu'ils possèdent toute l'information sur le mouvement des affaires. Dans cette optique, le bureau est censé offrir aux gestionnaires le système d'information nécessaire à la prise de décisions pertinentes et opportunes. Mais un gestionnaire s'intéresse assez peu au passé dont l'enregistrement correspond surtout à des obligations légales ou quasi légales. Les véritables besoins sont en avant sur le front des affaires.

On a alors pensé que l'introduction de l'informatique dans les organisations allait résoudre ce problème. Il n'en a rien été car l'ordinateur ne s'est intéressé qu'aux processus administratifs traditionnels et n'a pas repensé le bureau.

L'introduction de la bureautique a de réelles chances de sortir le bureau de son mode d'hier et d'offrir à chaque gestionnaire le système d'information qui lui est le mieux adapté. Grâce à de nouvelles technologies de plus en plus simples à utiliser et avec des interfaces conviviales, le gestionnaire pourra, s'il

le veut, accéder directement à l'information dont il a besoin. En plus, dans chaque information saisie, il pourra identifier ce qui est vraiment intéressant et cela sans "s'embarrasser des mille et une informations non pertinentes que la mécanique opaque de l'information lui dissimulait".[BLAS 82]

1.2 Objectif de notre travail

La bureautique, dont nous venons de faire une présentation très succinte, va constituer le cadre général du projet dans lequel s'insère notre mémoire. L'objectif de ce projet est de mettre au point un outil d'aide à l'automatisation du travail de bureau dans le cadre d'un atelier logiciel. Dans la suite de ce mémoire, les personnes chargées de l'analyse d'un bureau, seront référencées sous le nom d'*analyste*. Les personnes dont le travail va être automatisé par les analystes seront référencées sous le nom d'*utilisateur*.

Pour tout travail d'automatisation d'un bureau, les analystes qui en sont chargés procéderont toujours en étapes. Ces étapes, nous les décrivons en précisant pour chacune comment les analystes seront aidés par l'outil d'aide à l'automatisation.

Première étape : étude de l'existant

La toute première chose que doivent faire les analystes est généralement l'étude du bureau existant. Ils doivent étudier le comportement du personnel et à l'aide d'interviews, de questionnaires, d'observations directes, ils doivent établir en quoi consiste le travail à automatiser : quel est le travail réalisé, en quoi se décompose-t-il, quels sont les aspects du bureau concernés par ce travail, ... Les résultats de cette étude vont constituer ce que l'on a coutume d'appeler les spécifications du système à réaliser.

La mise au point de spécifications correctes est un problème difficile dû premièrement aux problèmes de communication entre l'analyste et l'utilisateur et deuxièmement aux ambiguïtés du langage naturel utilisé pour décrire ces spécifications. Ce problème se révèle d'autant plus important que toute la suite de l'analyse aura pour base les spécifications établies à cette première étape. Si celles-ci ne sont pas correctes, la suite ne le sera certainement pas !

L'atelier logiciel d'aide à l'automatisation se propose donc de mettre à la disposition de l'analyste un outil d'aide à la spécification.

A partir d'une modélisation du bureau, l'outil va mettre à la disposition de l'analyste un langage formalisé qui lui permettra d'exprimer ses spécifications tout en respectant le modèle du bureau développé. Les informations introduites, à l'aide d'un terminal, dans le système seront tout d'abord corrigées et ensuite mémorisées dans une base de données.

Cette base de données et les informations qui s'y trouvent serviront de point de départ à une simulation graphique du travail de bureau spécifié. Sous une forme animée le travail de l'utilisateur sera reproduit à l'écran à partir de la compréhension qu'en a eu l'analyste.

A partir de cette simulation va naître un processus de retour en arrière sur les spécifications de l'analyste à partir des corrections apportées par l'utilisateur.

Cette première étape pourrait donc se résumer comme suit :

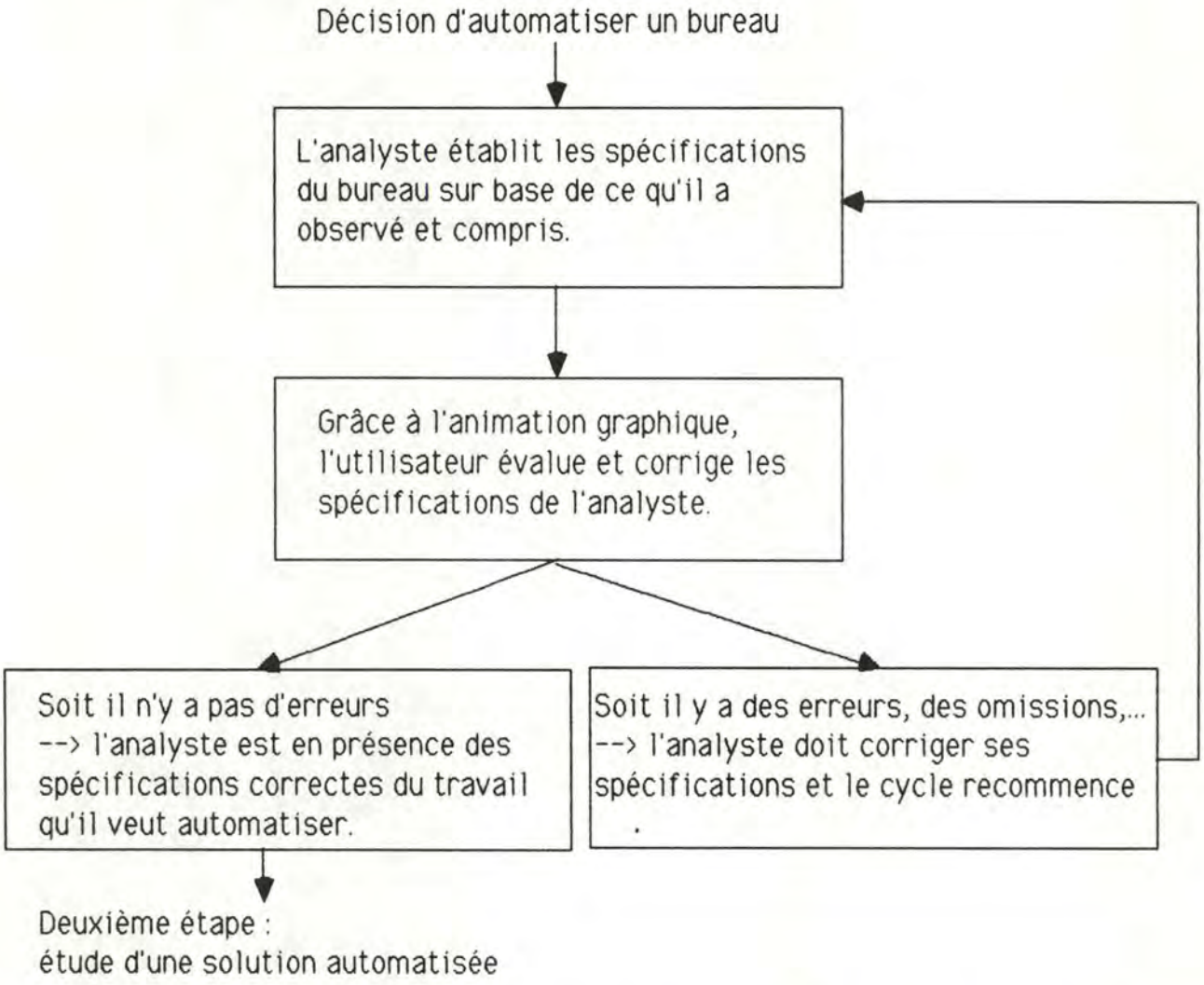


Fig. 1.1 : Processus d'itération pour la mise au point des spécifications

Deuxième étape : étude d'une solution automatisée

L'analyste, possédant des spécifications correctes, va pouvoir améliorer le travail de l'utilisateur en mettant à sa disposition de la technologie (ordinateur, imprimante, P.C., station de travail, modems, supports magnétique, ...). Il devra le faire en respectant certains critères comme le degré de l'automatisation désiré par les responsables, les possibilités techniques disponibles, le budget

disponible, la structure de l'organisation.

Lorsque l'analyste aura dressé une stratégie d'automatisation, il devra la proposer aux différents utilisateurs et responsables intéressés. Pour ce faire, deux possibilités lui sont offertes. Soit il fait l'investissement nécessaire pour acquérir le matériel et prend le risque de voir ses propositions refusées par les responsables, soit il leur fait un exposé théorique sur les modifications qui seront apportées au travail existant. Des deux solutions, la deuxième est la plus raisonnable car elle est la moins coûteuse dans le cas d'un retour en arrière. Pourtant ce n'est pas la meilleure car il y aura toujours le problème de communication entre l'analyste et l'utilisateur :

- l'analyste considère, dans son exposé qu'il fera aux utilisateurs, certains aspects informatiques comme étant évidents pour lui et ne les explique pas.
- les utilisateurs d'après ce qu'ils comprennent se font une fausse idée sur les possibilités exactes du futur système.

Ce manque de compréhension peut impliquer un désintérêt des utilisateurs face au système qui leur sera proposé car il ne produit pas du tout les résultats auxquels ils s'attendaient !

L'atelier logiciel se propose de résoudre ce problème en mettant à la disposition de l'analyste un langage formel lui permettant d'exprimer sa solution automatisée. A partir des informations reçues, le système fera à nouveau une simulation sous forme d'une animation graphique de la solution proposée.

L'utilisateur se trouve alors en présence de la simulation de ce que sera son futur travail et de ce qu'il devra faire. Il y aura donc moins d'ambiguïtés dans la compréhension et en connaissance de cause, l'utilisateur pourra faire ses remarques à l'analyste. Comme pour la première étape, un processus itératif va s'engager dont les principales étapes sont résumées à la figure 1.2.

Cette animation graphique pourra en outre servir d'outil d'apprentissage ou tout au moins comme moyen de familiarisation de l'utilisateur à l'ordinateur.

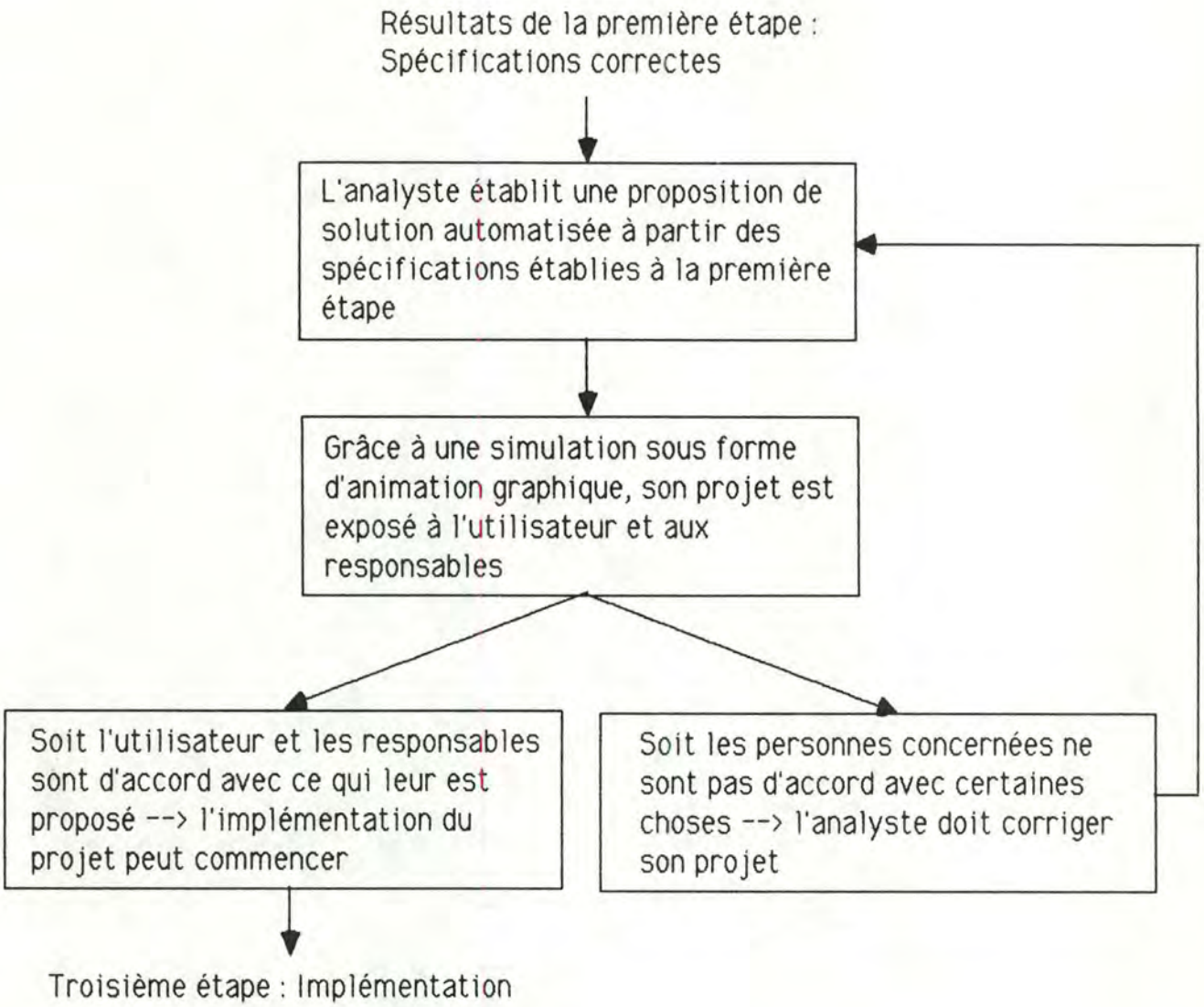


Fig. 1.2 : Processus itératif pour la mise au point d'une solution automatisée

Troisième étape : l'implémentation

L'analyste dispose désormais d'une solution d'automatisation du travail de bureau acceptée et comprise par les utilisateurs. Il peut donc commencer la réalisation pratique de son projet. Cela implique d'une part l'acquisition du matériel requis par la solution et d'autre part la mise sur pied des programmes qui vont constituer le logiciel permettant à l'utilisateur de faire son travail. Cela exige aussi la création du système d'information dans lequel seront mémorisées toutes les informations manipulées par l'utilisateur. La modélisation du bureau réalisée dans le cadre de l'atelier logiciel à la première étape pourra aider l'analyste à réaliser ce système d'information.

L'**originalité** même de cet atelier logiciel est l'utilisation d'une **interface graphique** comme outil de communication entre l'utilisateur et l'analyste. Cela est peu mis en oeuvre actuellement et donc la faisabilité d'un tel projet n'a jamais pu être réellement montrée. Une telle démonstration constitue l'objectif de notre mémoire qui va cependant se limiter à la première étape du projet. Pour

ce faire, nous allons réaliser un prototype de cette étape pour tenter de dégager les principales difficultés de ce projet, les problèmes à résoudre et surtout voir la faisabilité du projet à tout point de vue (logiciel et matériel).

Mais avant de décrire pourquoi et en quoi notre mémoire constitue un prototype, nous présentons au chapitre suivant une introduction générale sur la notion de prototype.

Chapitre 2
Le prototypage

CHAPITRE 2 : LE PROTOTYPAGE

2.1 Définition

La notion de prototype n'est pas réservée uniquement au domaine des logiciels, bien au contraire. Le prototypage est surtout employé dans le domaine de la fabrication d'objets ou de produits en grande série. Dans cette optique, on peut définir le prototype de la façon suivante :

> "Premier exemplaire construit industriellement d'un ensemble destiné à expérimenter en service les qualités de cet ensemble en vue de la construction en série". (définition extraite du Larousse).

Cette définition, nous pouvons la mettre en correspondance avec la définition du prototype dans le cas du développement d'un logiciel :

> "Le prototypage d'un système d'information a pour objectif la préparation de premières versions de ce système d'information (= le prototype), c'est-à-dire celles qui montrent les propriétés et les caractéristiques essentielles du système d'information futur (= le produit final ou le système final)." [ALAV 84]

Bien que ces deux contextes du prototypage aient globalement le même objectif, mettre au point un produit final à partir de prototypes, ils diffèrent cependant sur plusieurs points. Nous allons voir rapidement en quoi le prototypage d'un logiciel peut différer du prototypage industriel :

- le processus de prototypage logiciel mène à un produit unique au contraire du prototypage industriel qui vise la production en grande série.
- les caractéristiques désirées d'un logiciel ne sont pas toujours connues à l'avance; le prototype permettra de les affiner.
- le prototypage peut se faire sur une partie ou sur l'ensemble du système à concevoir.
- l'objectif est de mettre au point des démonstrations pratiques sur ordinateur de parties importantes du système final pour permettre d'améliorer les qualités du système.

2.2 La méthode du prototypage

La fabrication de logiciels par prototypage induit une démarche différente de celle impliquée dans le cycle conventionnel de développement d'un système d'information. Rappelons que le cycle de vie d'un système d'information (S.I.) (cfr. figure 2.1) se décompose globalement en quatre étapes : l'analyse fonctionnelle, le développement, l'implémentation et la maintenance. Un enchaînement linéaire avec de fréquents retours en arrière intervient entre ces

différentes étapes.

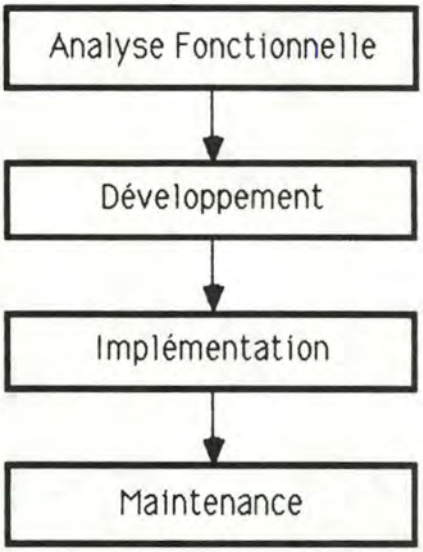


Fig. 2.1 : Méthode conventionnelle du cycle de vie.

Dans chacune de ces étapes, la méthode du prototypage peut être utilisée pour aider à concevoir un travail respectant mieux les spécifications du problème (cfr. figure 2.2). Dans la maintenance, la méthode du prototypage peut y être introduite en considérant que le logiciel que l'on corrige est lui-même un prototype de la nouvelle version que l'on réalise. Avec cette vision des choses, on pourrait se demander si la maintenance en elle-même n'est pas du prototypage ?

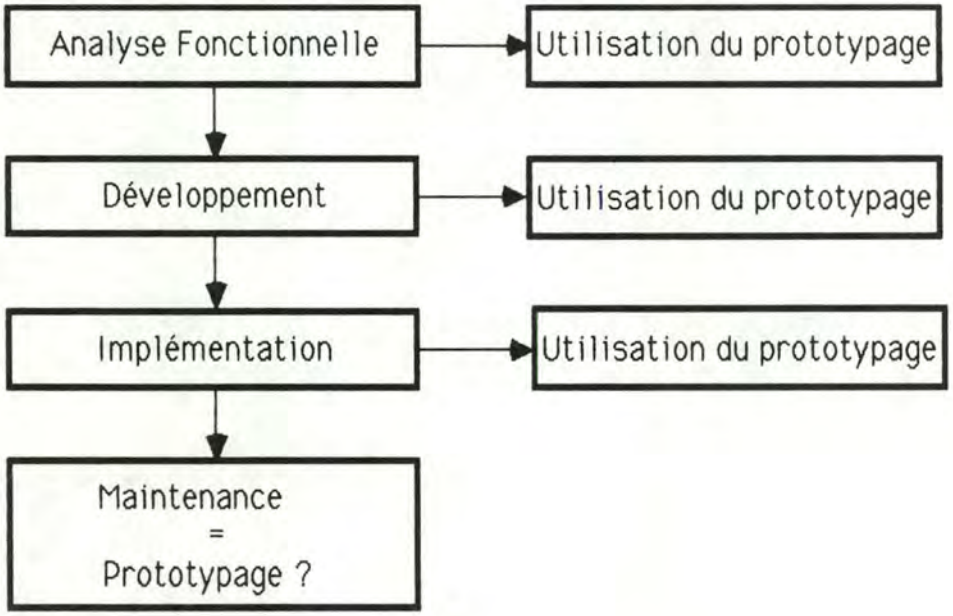


Fig. 2.2 : Introduction de la méthode du prototypage dans le cycle de vie d'un logiciel

La méthode du prototypage (cfr. figure 2.3) peut elle aussi être subdivisée en quatre étapes [BUDD 84], [CANN 81] : la sélection fonctionnelle, la construction, l'évaluation et le choix de l'usage futur du prototype. Entre ces

étapes, un enchaînement linéaire intervient de la même manière que dans la méthode précédente mais il peut être interrompu par un retour en arrière à la troisième étape. Nous verrons plus loin dans quel cas.

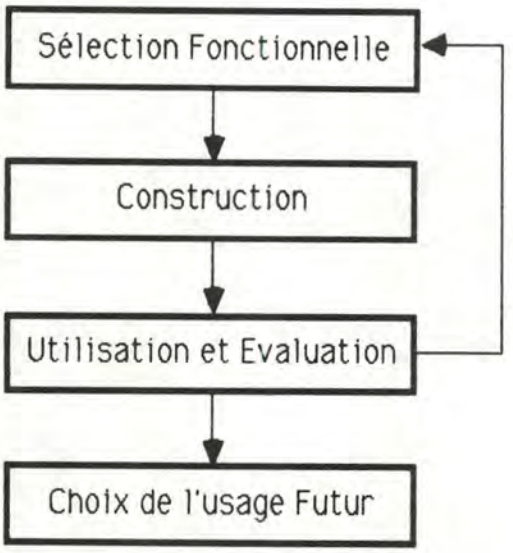


Fig. 2.3 : Méthode du prototypage.

Première étape : la sélection fonctionnelle

La sélection fonctionnnelle se rapporte au choix des fonctions que le prototype sera sensé exposer. Cette sélection doit toujours être basée sur des tâches de travail significatives et qui pourront servir de modèle lors de l'utilisation de celui-ci. L'étendue des fonctions offertes par le prototype n'est jamais la même que celle du produit final bien qu'en théorie, elle pourrait l'être. Cette différence d'étendue des fonctions peut se résumer en deux propositions :

- seules les fonctions considérées comme cruciales sont implémentées totalement dans le prototype. Il s'agit du prototypage vertical.
- toutes les fonctions du système final se retrouvent dans le prototype mais sous forme réduite c-à-d qu'une de leur partie peut être omise ou éventuellement simulée. Il s'agit du prototypage horizontal.

Souvent, ces deux éléments sont combinés dans les différentes parties du prototype.

Deuxième étape : la construction

La construction se rapporte à l'effort fait pour disposer effectivement du prototype. Cet effort sera moins important que celui fournit pour le système final grâce à des techniques et à des outils appropriés à la construction de prototypes. On pourra consulter [LAPA 86] pour plus de détails à ce sujet.

Dans la construction du prototype, l'accent doit être mis sur le caractère

évaluable du prototype plutôt que sur son usage à long terme. Cela implique que certains aspects du produit final (performances, droits d'accés, interblocage,...) peuvent être omis, à moins qu'ils ne fassent partie des finalités du problème. Par exemple, dans un logiciel de transactions bancaires, l'aspect sécurité ne peut être négligé.

Troisième étape : l'évaluation

L'évaluation doit être considérée comme l'étape décisive dans la méthode du prototypage car elle fournit un élément de communication entre utilisateurs et concepteurs pour la suite du processus de développement. C'est pourquoi, le chef de projet doit s'assurer que toutes les ressources nécessaires sont disponibles pour la meilleure évaluation possible, notamment la participation de tous les groupes significatifs parmi les utilisateurs futurs. L'évaluation devra être faite à partir d'un document décrivant les critères d'évaluation et spécifiant quel travail doit accomplir le prototype.

En fonction du résultat de l'évaluation, soit on recommencera les trois premières étapes, en y apportant les améliorations nécessaires afin d'arriver à une évaluation positive, soit on décide de l'usage futur du prototype.

Quatrième étape : choix de l'usage futur

Deux choix sont possibles pour l'usage futur du prototype. Soit le prototype servira éventuellement d'outil d'apprentissage pour l'ensemble des futurs usagers et sera ensuite jeté, soit le prototype sera utilisé partiellement ou complètement comme composant du système final.

Comme signalé à la figure 2.2, la méthode du prototypage et la méthode du cycle de vie ne sont ni concurrentes ni exclusives. Un prototype peut être utilisé dans chacune des étapes du cycle de vie. En fonction donc, des buts que se fixe le prototypage, nous aurons une approche différente de celui-ci. C'est ce que le point suivant va exposer.

2.3. Les approches du prototypage

Dépendant des objectifs que l'on désire atteindre, nous pouvons distinguer trois grandes classes de prototypage. Pour chacune, nous en ferons une description théorique [BUDD 84] et ensuite, dans la mesure du possible, nous donnerons l'exemple d'un logiciel existant pouvant être classé dans cette classe. Il faut cependant remarquer que les frontières entre ces classes ne sauraient être clairement établies car ces classes se recouvrent partiellement.

2.3.1 Le prototypage exploratoire

Cette approche se centralise sur les problèmes de communication entre le

concepteur du logiciel et l'utilisateur, particulièrement dans les premières étapes de développement d'un logiciel. Ces problèmes résultent du fait que d'une part les concepteurs possèdent peu de connaissance à propos de l'application et que d'autre part les utilisateurs n'ont pas une idée claire des possibilités exactes de l'ordinateur

Pour pallier ces problèmes, une démonstration pratique des fonctions du système peut servir à se faire une bonne idée du système et à promouvoir une coopération créative entre toutes les parties impliquées. En outre, cette démonstration ne devrait pas se focaliser sur une solution en particulier mais essayer d'envisager l'ensemble des solutions alternatives qui méritent d'être discutées.

L'utilisateur sera profondément influencé par cette démonstration au point de vue de ses attentes. C'est pourquoi, le choix des fonctions implémentées, leurs caractéristiques et leurs relations avec le reste du système devront faire l'objet d'une attention toute particulière. Par exemple, il serait utile de commencer avec un ensemble de fonctions permettant à l'utilisateur d'exécuter une de ses tâches complètement avec l'aide du prototype. Il pourrait ainsi juger du caractère réalisable de la tâche exécutée et il pourrait, en procédant par analogie, voir comment les autres tâches s'exécuteront.

Il doit être clair au concepteur que l'objectif de cette approche est d'améliorer les spécifications du système final et qu'il n'y a aucun engagement à voir le prototype reproduit tel quel dans le système final. C'est en arrivant à avoir le maximum de compréhension vis-à-vis des exigences de l'utilisateur que l'on arrivera à mettre au point les spécifications du système final via le prototype.

Ce prototypage exploratoire est compatible avec l'approche normale du développement d'un logiciel et sert à améliorer la première étape : l'analyse fonctionnelle.

De façon concrète, il y a des logiciels qui peuvent s'apparenter avec la notion de prototypage exploratoire . Tel est le cas, par exemple, du logiciel DSL-PROTO qui est inclus dans l'atelier logiciel IDA (Interactive Design Approach). [BOPI 83]

Dans le cadre de toute analyse fonctionnelle d'un problème à résoudre, l'analyste qui en est chargé est amené à établir les spécifications de ce problème et les spécifications du futur système. Après les avoir exprimées dans un formalisme défini par le langage DSL (Dynamic Specification Language), le logiciel DSL-PROTO va construire une maquette/prototype chargée de montrer à l'analyste et à l'utilisateur la solution proposée. L'exécution de ce prototype doit essentiellement leur permettre de vérifier si les règles de traitement et les enchaînements qui ont été spécifiés produisent les résultats qu'ils en

attendent. Par le processus de retour en arrière exposé dans la méthode du prototypage, le prototype permettra à l'analyste d'améliorer les spécifications.

2.3.2 Le prototypage expérimental

Dans l'approche du prototype expérimental, l'accent est mis sur la vérification que la solution proposée est adéquate avant d'investir dans une implémentation à grande échelle du système final.

Dans le cycle normal de la conception d'un logiciel, le prototypage expérimental est approprié dans n'importe quelle phase dès que les spécifications initiales ont été écrites.

Ce genre de prototypage a plusieurs domaines d'application comme la transparence de l'interface homme-machine, l'acceptabilité des performaces proposées pour un système ou la possibilité de développer un logiciel suivant les ressources disponibles.

Nous explorons maintenant quelques stratégies d'application du prototypage expérimental.

a) La simulation fonctionnelle complète

La simulation fonctionnelle complète montre toutes les fonctions du système final qui devront être disponibles pour son utilisation normale par les utilisateurs. Le prototypage peut être construit avec des techniques qui offrent des facilités d'implémentation et de modification plutôt que l'optimalisation des performances du système. On pourrait qualifier cette approche de prototypage horizontal.

b) La simulation d'interface

La simulation d'interface présente aux utilisateurs une interface homme-machine qui offre toutes les caractéristiques de l'interface finale mais ne gère que cette interface. L'utilisateur voit donc à quoi ressemblera le système final mais il n'y a aucune validation des informations qu'il serait amené à introduire et l'exécution du programme a lieu sans la présence de données réelles.

Comme logiciel permettant de faire de la simulation d'interface, nous pouvons citer OMEGA qui est un atelier intégré pour la production et la maitenance d'applications transactionnelles. OMEGA fournit, notamment, une fonction de maquettage/prototypage qui permet, sur le poste de travail du concepteur ou dans l'environnement d'utilisation de l'application future, une validation précoce et poussée de la définition fonctionnelle, notamment des dialogues. Pour plus d'informations sur OMEGA, le lecteur interessé pourra se

référer à [META 86].

c) La programmation squelettique

La programmation squelettique expose aux utilisateurs la struture générale du système sur la base de quelques fonctions sélectionnées comme étant représentatives des autres fonctions. Les fonctions qui sont choisies doivent permettre à l'utilisateur :

- d'exécuter complètement quelques-unes de ses tâches de travail,
- de dresser une analogie avec le reste du travail qui n'est pas pris en considération par le prototype.

Cette forme de prototypage est intéressante pour montrer comment le système sera inclus dans le cadre général du travail de l'utilisateur et pour montrer les relations entre le travail réalisé manuellement et celui réalisé à l'aide du système. Il peut également servir à simuler les performances attendues du système final et donc à se faire une bonne idée de ce qui est acceptable ou non par l'utilisateur. On pourrait qualifier cette approche de prototypage vertical.

d) Simulation fonctionnelle partielle

La simulation fonctionnelle partielle est utilisée simplement pour tester une hypothèse à propos du système. Par exemple, pour voir si un algorithme proposé produira des résultats acceptables dans un nombre suffisant de cas en utilisant une quantité raisonnable de ressources.

Pour terminer cette approche et pour illustrer le prototypage expérimental, on peut citer le logiciel DSL-SIM qui est intégré dans l'atelier logiciel IDA. Comme nous l'avons signalé ci-dessus, le prototypage expérimental peut être réalisé lorsque les spécifications du système sont établies.

Il peut toutefois se révéler intéressant d'évaluer si une solution conceptuelle est réalisable par rapport aux ressources dont dispose l'organisation. C'est pourquoi à partir de la spécification des traitements et de la spécification des ressources, DSL-SIM va générer des programmes qui vont simuler le fonctionnement du sytème. Cette simulation va déboucher sur une éventuelle modification des spécifications des traitements et des ressources si les résultats de DSL-SIM révèlent que le système ne respecte pas les critères de performances demandées. Il y a donc un processus de retour en arrière qui va améliorer et corriger les spécifications du système.

Si nous devions classifier DSL-SIM dans les stratégies d'application du prototypage expérimental, on dirait qu'il s'agit par certains de ses aspects d'une part d'une simulation fonctionnelle complète (toutes les fonctions du

système sont évaluées du point de vue de leurs performances) et d'autre part d'une programmation squelettique (on optimalise les performances et on ne s'attache qu'aux traitements et aux ressources).

2.3.3 Le prototypage évolutif

Dans le prototypage évolutif, le produit lui-même est vu comme une série de versions et chaque version peut être évaluée et servir de prototype pour ses successeurs.

Ce prototypage évolutif est l'approche la plus éloignée, et la plus controversée, de la signification du prototype. Certains auteurs, plutôt que *prototypage*, parleraient plutôt en terme de *développement en versions*.

Cette approche a tout de même été classée dans le prototypage parce que la frontière entre le prototypage évolutif et les autres formes du prototypage ne sont pas claires (tout prototype permet la révision des exigences de départ et donc contribue à l'évolution du système). Ensuite, les prototypages exploratoires et expérimentaux peuvent être appliqués dans les premières étapes d'une stratégie évolutive.

Le prototypage évolutif peut s'appliquer lorsque :

- l'organisation, dans laquelle se trouve installé le système, évolue et donc de nouvelles exigences émergent pour le système.
- après utilisation, le système laisse apparaître des améliorations possibles et donc de nouvelles exigences surviennent.

Le prototypage évolutif rompt l'ordre linéaire des phases de l'approche traditionnelle de développement de logiciels au profit de cycles de développement successifs.

2.4 Principaux avantages et dangers du prototypage

Le prototypage a pour avantage, selon l'approche utilisée, de clarifier les exigences de l'utilisateur, de valider les spécifications, de vérifier l'adéquation d'une solution proposée par rapport aux exigences ou d'adapter un système à ses changements de contexte d'utilisation.

Le danger du prototypage vis-à-vis des spécifications est qu'il conduise à un manque de soin dans l'élaboration de celles-ci. Le prototype ne doit pas se faire au détriment d'une analyse rigoureuse au risque d'entrer dans un jeu d'essais-erreurs qui finira par lasser l'utilisateur.

La méthode du prototypage n'apparaît également n'être envisageable que lorsque les deux parties sont bien informées sur cette approche. L'utilisateur

pour qu'il ne se lasse pas dans le cycle d'évaluation des prototypes et le concepteur pour qu'il ne se lasse pas devant les changements fréquents dans les exigences de l'utilisateur du prototype.

2.5 Présentation du mémoire en tant que prototype

Un des buts de notre travail est d'établir la *faisabilité* de la première étape du projet général (cfr. point 1.2). Nous examinons maintenant pourquoi le prototypage semble être une bonne méthodologie vis-à-vis de cet objectif et sous quelle forme le prototypage sera utilisé.

Tout d'abord, il convient de vérifier si le genre d'animation graphique que nous proposons (déplacement d'icônes en deux dimensions) est assez explicite ou non pour l'utilisateur dans la simulation de sa tâche. Il faudra voir si les outils graphiques utilisés sont assez puissants et par exemple, voir s'il ne sera pas nécessaire de faire une animation graphique en trois dimensions. Dans cette optique, nous pouvons dire que notre travail sera un prototype expérimental et plus précisement un *prototype expérimental d'interfaces*.

Ensuite, afin de pouvoir vérifier ce premier aspect des choses, nous choisissons de simuler *quelques* tâches de bureau. Notre travail n'a pas la prétention de simuler toutes les tâches de bureau. Il est cependant nécessaire que la modélisation du bureau adoptée ne soit *pas incompatible* aux tâches non supportées par le prototype et qu'elle reste *ouverte* à celles-ci. A partir des tâches simulables graphiquement par notre prototype, l'utilisateur devra pouvoir établir des analogies avec les tâches que notre prototype ne peut simuler. Dans cette optique, nous pouvons dire que notre travail sera un prototype expérimental et plus précisement un *prototype expérimental à programmation squelletique*.

2.6 Conclusion

En conclusion de ces deux premiers chapitres, nous dirons simplement que le projet visant à établir les spécifications du travail de bureau à l'aide d'une interface graphique relève de la méthode du prototypage. En effet, l'analyste utilisera le simulateur graphique pour pallier aux problèmes de compréhension qu'il risque de rencontrer avec l'utilisateur. Nous pourrions qualifié la première étape du projet de prototypage exploratoire.

Dans ce cadre, nous réalisons nous même un prototype visant justement à montrer la faisabilité de ce projet. Dès lors, pourrions nous parler de méta-prototype ?

Chapitre 3
La tâche de bureau

CHAPITRE 3 : LA TACHE DE BUREAU

3.1 Introduction générale au concept de tâche

Notre travail de recherche portant sur le développement d'un outil de description et d'animation graphique des tâches de bureau, il convient de décrire d'abord ce que nous entendons par bureau et d'extraire de cette description les principaux concepts.

Notre description se fera en deux étapes distinctes. La première exposera brièvement un modèle général des organisations tandis que la deuxième appliquera ce modèle au cas précis qui nous occupe.

3.1.1 Modèle général des organisations

Toute organisation peut être décrite au moyen du modèle des organisations de Leavitt [LEAV 65]. Dans ce modèle, Leavitt décrit l'organisation comme un graphe dont les éléments constitutifs (tâche, technologie, personne et structure) sont en interaction et mutuellement ajustés.

Pour plus de détails sur cet équilibre et son influence sur la vie de l'organisation, on pourra consulter [KEEN 81].

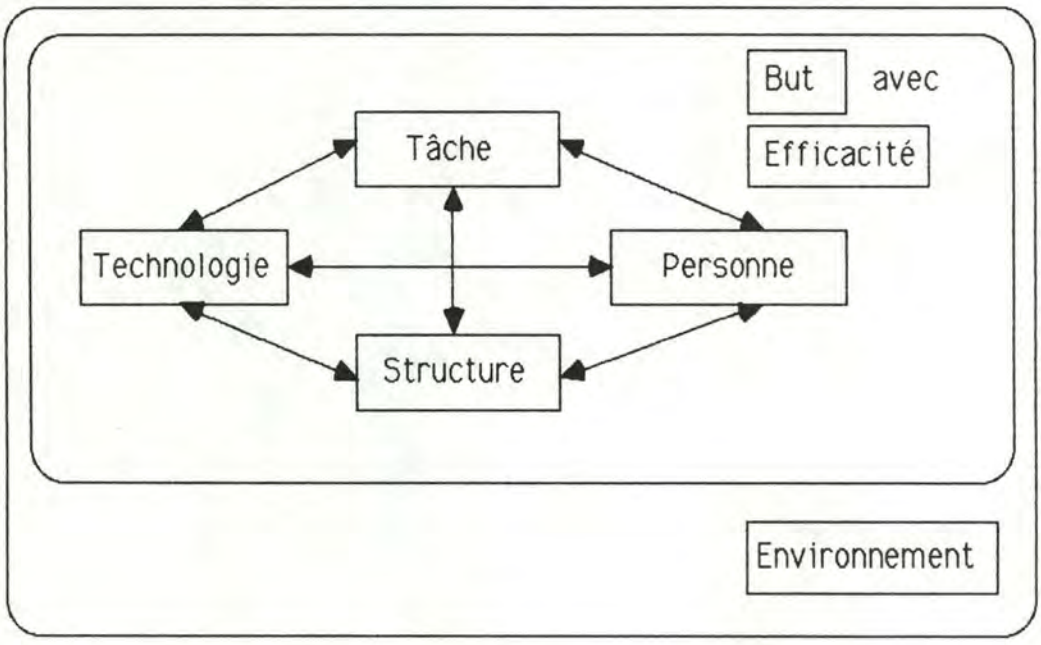


Fig. 3.1 : Modèle des organisations de Leavitt.

L'interprétation de ce modèle (cfr. figure 3.1) est la suivante :

- l'organisation est une entité sociale dans laquelle un ensemble de *personnes* sont réunies pour poursuivre un même *but*.

- la réalisation de ce but doit répondre à certains critères d'*efficacité* mesurable.

- afin d'atteindre ce but, des personnes exécutent différentes *tâches* en utilisant le cas échéant de la *technologie*.

- personne, tâche et technologie sont intégrées dans une *structure* qui détermine les relations entre les tâches, les responsabilités et le pouvoir des personnes et ainsi, permet à l'organisation de fonctionner.

- enfin, l'organisation dépend de l'*environnement* avec lequel elle est en relation.

Avec Hines [HINE 85], nous conviendrons que la tâche est un travail discernable impliquant des activités, des ressources et des produits caractéristiques.

Le produit d'une tâche peut servir comme mesure de la performance de ceux qui l'exécutent. Par exemple, le nombre de factures produit peut être la mesure d'une tâche de facturation.

Pour la définition du terme de ressource, le lecteur intéressé pourra se référer à [BOPI 83] et à [DACH 86] où il trouvera une modélisation complète des ressources dans une organisation.

3.1.2 Modèle général des organisations appliqué au bureau

Reprenons maintenant ce modèle et appliquons-le au bureau, puisque celui-ci peut être considéré comme une organisation réduite, et examinons-en chacun de ses composants (cfr. figure 3.2).

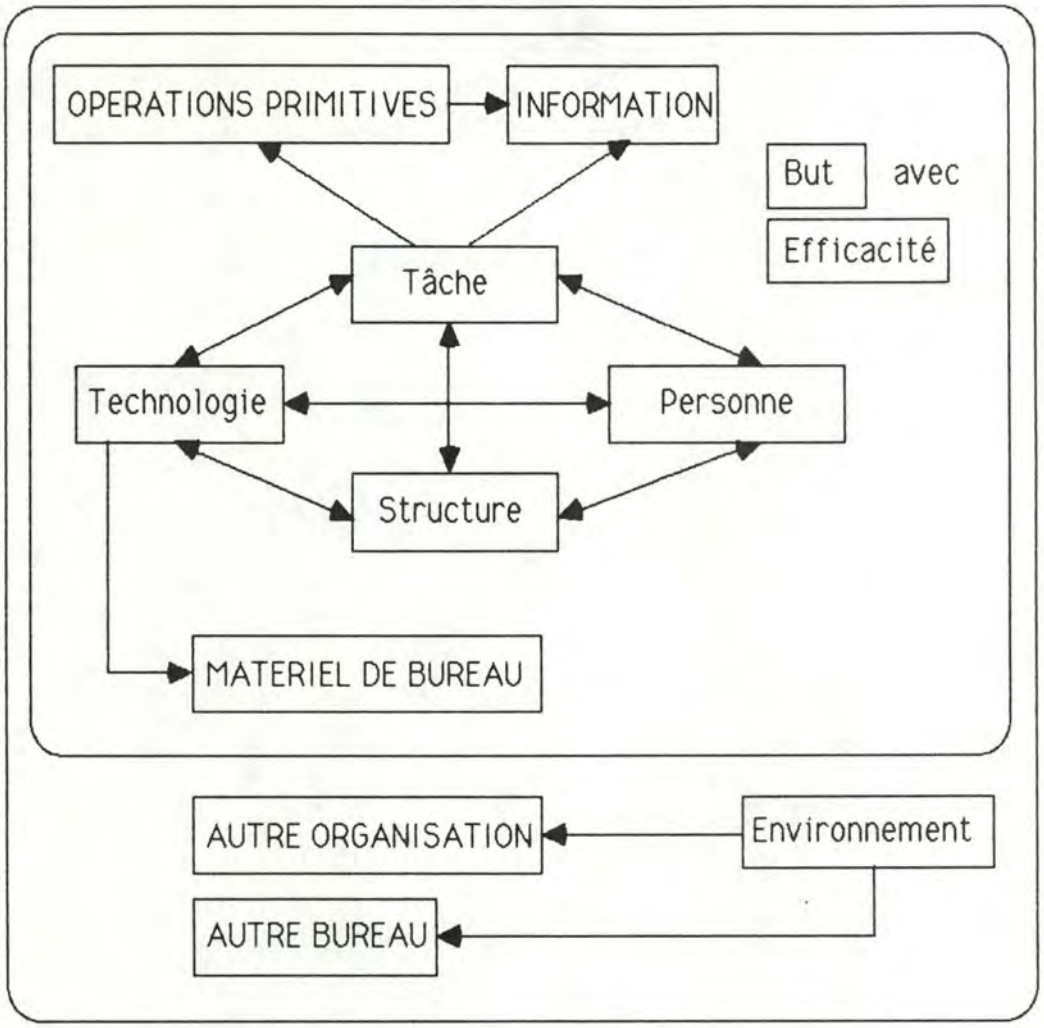


Fig. 3.2 : Modèle des organisations de Leavitt appliqué au BUREAU

Les seules tâches dont nous nous occuperons sont celles qui traitent de l'information et nous définirons le concept d'information comme un ensemble structuré de signes ou de symboles porteurs de connaissance pour celui qui les reçoit.

Nous poserons également comme hypothèse qu'une tâche de bureau obéit à un minimum de structure c-à-d que nous ne traiterons que des tâches structurées et semi-structurées [KEEN 78] et donc qu'elles peuvent se décomposer en opérations primitives portant sur les différentes formes que prendra l'information dans le bureau.

Si nous examinons maintenant l'élément technologie du graphe, on peut dire que dans notre étude, cela correspond au matériel de bureau.

Les personnes quant à elles, interviennent dans le bureau non seulement en tant qu'élément de la structure de l'organisation mais également comme facteur de production pour l'exécution des tâches.

L'environnement du bureau peut être soit un autre bureau de l'organisation dont il fait partie, soit un autre bureau d'une autre organisation soit encore, une autre organisation à part entière.

3.2 L'approche par objets pour la description du bureau

3.2.1 Introduction

Comme nous l'avons vu au point ci-dessus, un bureau est un ensemble de concepts tels que de la technologie, de l'information, ... ayant des relations entre eux. Le schéma qui représente ces concepts et ces relations a été donné à la figure 3.2 et constitue un premier modèle du bureau c'est-à-dire une première représentation aussi fidèle que possible du réel perçu.

Nous affinons maintenant cette représentation du réel pour arriver à un modèle du bureau suffisamment précis afin de pouvoir simuler l'exécution d'une tâche à partir de celui-ci.

Comme nous le verrons, cette analyse va nous mener à dégager un certain nombre de concepts que l'on va décrire et structurer dans l'optique des langages orientés objets. Ensuite, nous adopterons un modèle de réprésentation dans lequel nous transcrirons les concepts du bureau identifiés.

3.2.2 L'approche par objets

Lorsque l'on désire percevoir ou analyser un aspect du monde réel, on a tendance, dans un premier temps, à le regarder de l'*extérieur*. Ainsi, lorsque l'on étudie un objet, on commence par regarder ses caractéristiques (par exemple, comment peut-on l'utiliser ou que peut-on en faire ?) et dans le cas où une analyse très précise est nécessaire, on regarde l'*intérieur* de l'objet (par exemple, comment l'objet est-il fait et de quoi se compose t-il ?) [RENT 82]. Cette façon d'aborder le réel est une caractéristique essentielle des langages orientés objets et c'est celle-ci que nous utilisons pour décrire un bureau.

Avant de réaliser l'analyse du bureau, nous introduisons le fonctionnement global d'un langage orienté objets. Plus particulièrement, c'est la structuration des données plutôt que leur traitement qui nous intéressera . Le langage qui nous servira de référence pour cette introduction est le langage SMALLTALK-80 car celui-ci est reconnu comme le modèle de référence des langages orientés objets.

3.3 Introduction aux langages orientés objets [ROBS 81], [ROGO 83]

3.3.1 Les notions d'objet, d'information et de protocole

Les langages orientés objets travaillent sur un élément principal qui est l'objet. L'objet a la particularité d'être constitué à la fois par une information qui lui est propre et par l'ensemble des opérations que l'on peut appliquer à cet objet. Cet ensemble d'opérations va constituer le protocole de l'objet ou encore son interface avec l'extérieur (on regarde un objet de l'*extérieur*). Par exemple, si on se réfère à une application chargée de gérer l'affichage de fenêtres sur l'écran d'un terminal, on pourrait considérer que les fenêtres sont des objets. L'interface d'un objet serait constituée par l'ensemble des opérations que l'on peut lui appliquer : OUVRIR, AGRANDIR, BOUGER, ...

3.3.2 La notion de classe

Lorsque l'on travaille avec un langage orienté objets, tout est considéré comme objet. Par exemple, le chiffre 3 est un objet dont l'interface est constituée par les opérateurs algébriques. C'est pourquoi, on va pouvoir rassembler tous les objets qui sont identiques dans des classes. Une classe est une description de un ou de plusieurs objets similaires.

Tous les objets appartenant à une même classe doivent avoir le même type d'information et posséder la même interface. Le type des informations propre aux objets de la classe et l'interface des objets seront précisés dans la description de la classe.

Par exemple : - 3 fait partie de la classe ENTIER,
- une fenêtre fait partie de la classe FENETRE dont la description est composée de :

    - l'information relative aux fenêtres constituée par deux paires de coordonnées (x1,y1) et (x2,y2); elles donnent les coordonnées du coin supérieur gauche et du coin inférieur droit.
    - l'interface = {BOUGER, DETRUIRE, REDUIRE, AGRANDIR, CREER}.

Chaque fenêtre possède donc comme information deux paires de coordonnées mais pour celles-ci, chaque fenêtre peut avoir des valeurs différentes : par exemple, si deux fenêtres se trouvent à des endroits différents sur l'écran. La schématisation des concepts de classe et d'objet est donnée à la figure 3.3.

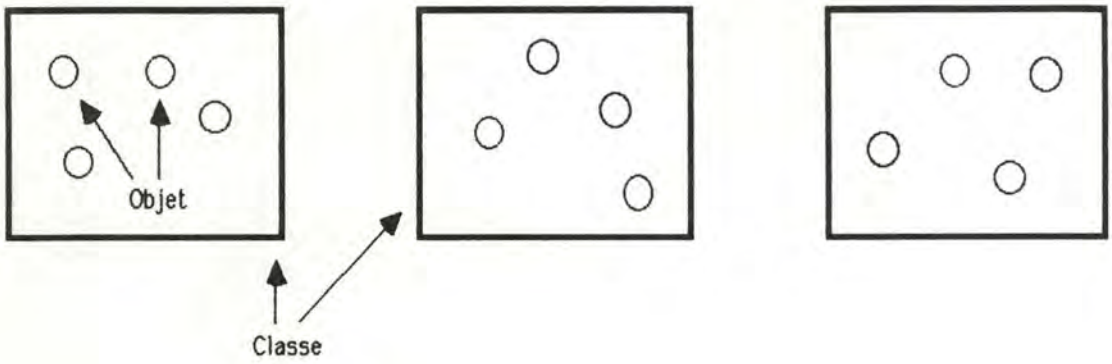


Fig. 3.3 : Représentation d'une classe

3.3.3 Les notions de super-classe et de sous-classe

Maintenant à l'intérieur d'une classe, il se peut qu'un sous-ensemble d'objets présente des propriétés communes. Ce sous-ensemble présente donc des propriétés qui lui sont propres et par conséquent apporte un complément de précision à la définition de la classe à laquelle il appartient. Ce supplément de précision ne se justifiant que pour une partie des objets d'une classe, nous les regrouperons en une classe qui aura la qualification de sous-classe. La classe à laquelle appartient cette sous-classe portera quant à elle la qualification de super-classe.

Par exemple, dans la classe FENETRE, on pourrait supposer qu'il y a certaines fenêtres dans lesquelles on peut écrire et d'autres pas. On va alors former une sous-classe de FENETRE dont le nom est FENETRE-ECRITURE et qui rassemble les fenêtres dans lesquelles on peut écrire. La description de la sous-classe FENETRE-ECRITURE pourrait être :

- information = vide; c'est-à-dire que les fenêtres où l'on peut écrire n'ont pas d'information particulière.
- l'interface = (ECRIRE, EFFACER).

Les notions de super-classe et de sous-classe permettent d'introduire le principe d'abstraction sur les données. Une abstraction sur les données permet de cacher de l'information à certaines applications. Par exemple, on pourrait supposer une application qui ne s'intéresse pas aux fenêtres dans lesquelles on peut écrire. Dans ce cas, cette application fera abstraction de la sous-classe FENETRE-ECRITURE. Pour chaque application, on peut ainsi grâce à ces concepts choisir le niveau de précision des informations sur lequel on veut qu'elle travaille.

3.3.4 La notion d'héritage de propriétés [CARB 81]

Le principe d'héritage en application dans les langages objets stipule tout simplement que tout objet appartenant à une sous-classe hérite des propriétés de sa super-classe ; il s'agit donc de l'héritage des informations et de l'interface de la super-classe.

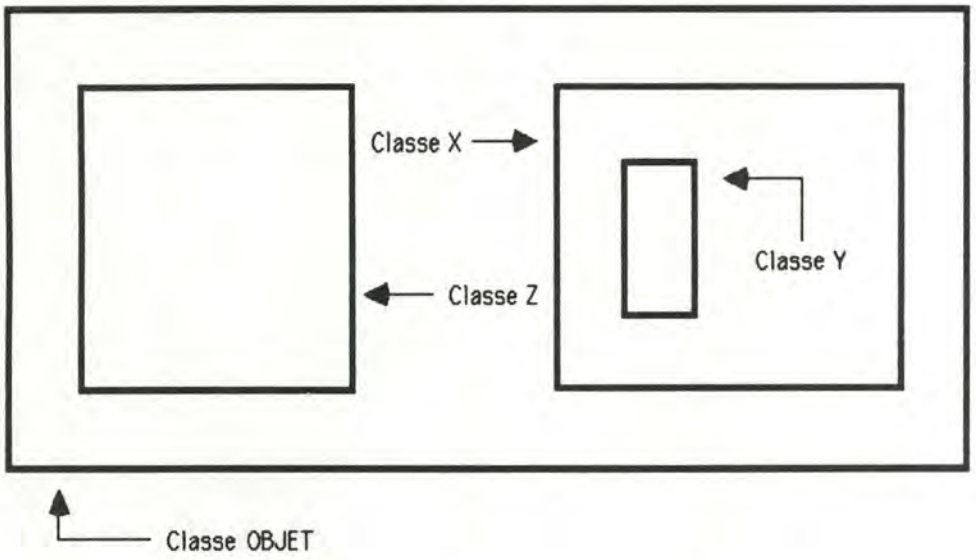
Par exemple, si un objet appartient à la classe FENETRE-ECRITURE alors, on peut dire qu'il s'agit d'une fenêtre mais dans laquelle on peut écrire; elle hérite ainsi des propriétés de la classe FENETRE. Pour la classe FENETRE-ECRITURE, sa description sera composée de :

- deux paires de coordonnées (héritage de la classe FENETRE).
- l'interface = {BOUGER, DETRUIRE, REDUIRE, AGRANDIR, CREER} (héritage de la classe FENETRE) + {ECRIRE, EFFACER} (son interface).

3.3.5 Quelques propriétés

- tout objet appartient à une et une seule classe. Il faut toutefois remarquer que ceci est une particularité de SMALLTALK-80; en règle générale, tout objet peut appartenir à une ou plusieurs classes,
- toute classe appartient à une super-classe sauf une qui est la classe système OBJET et qui englobe toutes les classes définies,
- une classe peut avoir plusieurs sous-classes,
- une classe ne peut être sous-classe que d'une et une seule super-classe. Il faut toutefois remarquer que ceci est une particularité de SMALLTALK-80; en règle générale, une classe peut être sous-classe d'une ou plusieurs autres classes,
- une super-classe ou une sous-classe est aussi une classe,
- une classe est un objet,
- un objet d'une classe hérite des propriétés de sa super-classe et de toutes les super-classes qui peuvent encore se trouver à un niveau supérieur.

Un exemple de hiérarchie de classes est donné à la figure 3.4.



La classe y est une sous-classe de la classe x
La classe x est la super-classe de la classe y
La classe x est une sous-classe de la classe OBJET
La classe z est une sous-classe de la classe OBJET
La classe OBJET est la super-classe des classes x et y

Fig. 3.4: Hiérarchie des classes

3.3.6 L'exécution d'un programme

Ce que nous avons décrit jusqu'ici ne concerne que la représentation des données. Cet aspect étant le seul qui nous intéresse pour le moment, nous ne décrivons que très brièvement le principe d'exécution d'un programme dans SMALLTALK-80. Le lecteur intéressé pourra consulter [ROGO 83] et [XERO 81] pour plus de précision à ce sujet.

Dans SMALLTALK-80, l'information est manipulée par l'envoi d'un message à l'objet représentant l'information. Tout message contient le nom de l'objet récepteur, l'identification de l'opération à effectuer sur l'objet et peut contenir un ou plusieurs arguments. La caractéristique principale du message est qu'il permet au programmeur de dire qu'il veut exécuter quelque chose sur un objet sans devoir dire comment. Pour manipuler l'information d'un objet, le programmeur doit seulement connaître l'interface de l'objet; interface qui est décrite dans la classe de l'objet. Cette caractéristique introduit donc un deuxième type d'abstraction cette fois-ci non plus sur l'information mais sur les traitements.

Lorsqu'un objet reçoit un message, il cherche l'opération de son interface qui s'identifie avec l'opération qui se trouve mentionnée dans le message reçu. Quand la correspondance a réussi, l'opération est exécutée sur l'objet. Toute opération se trouvant dans une interface se trouve associée à un ensemble d'actions; cet ensemble forme ce qu'on appele une méthode. Lorsque la corres-

pondance a réussi, c'est la méthode associée à l'opération qui est exécutée. Dans cette méthode, on peut trouver notamment des envois de messages vers d'autres objets.

3.3.7 Vue extérieure et vue intérieure d'un objet

Dans l'introduction, nous avons fait référence à la façon de percevoir le réel. Maintenant que les principaux concepts de SMALLTALK-80 sont définis, nous pouvons préciser ce que cela veut dire dans le cadre d'un langage orienté objets. Regarder un objet de *l'extérieur* veut dire prendre connaissance de son protocole. Tandis que regarder *l'intérieur* d'un objet signifie regarder comment les méthodes sont construites.

3.4 Modélisation du bureau dans l'optique d'une analyse par objets

3.4.1 Introduction

Sur la base de la classification et de la hiérarchisation des objets en classe, sous-classe et super-classe, nous allons analyser le bureau.

Pour réaliser cette analyse, nous utiliserons tout d'abord les travaux de N. Dachouffe [DACH 86] qui a réalisé un modèle d'information du bureau; elle y a étudié les supports d'information dans un bureau.

Nous utiliserons aussi les travaux de S. Robert [ROBE 86] qui a réalisé une modélisation des objets de rangement dans un bureau.

Devant réaliser une simulation d'une tâche de bureau, nous serons amenés à considérer en plus des objets de rangement et des objets informationnels d'autres classes d'objets. Comme nous l'expliquerons, plus tard dans ce chapitre, nous obtiendrons quatre classes d'objets :

- classe objet informationnel [DACH 86],
- classe objet de rangement [ROBE 86],
- classe objet d'interface,
- classe outil de travail.

Avant de décrire chacune de ces classes en détail, nous pouvons en nous référant au modèle du bureau (cfr. figure 3.2), dire que ces classes d'objets représentent deux choses :

- l'information véhiculée par une tâche et qui est représentée par les objets informationnels.
- la technologie que la tâche utilise pour s'exécuter et qui est représentée par les objets de rangement, d'interface et par les outils de travail.

Toutes deux jouent un rôle primordial dans le bon déroulement d'une tâche.

Il faut préciser que pour cette analyse du bureau, nous n'avons jamais été guidé par la description d'une tâche en particulier. Nous avons essayé d'établir un modèle qui soit le plus proche possible de la réalité pour qu'il puisse nous servir d'environnement à la simulation de tâches quelconques. Cependant, il faut bien reconnaître que la description d'un modèle de bureau suffisamment complet pour permettre la simulation de toutes les tâches de bureau est impossible; pour la simple raison, qu'il est impossible d'établir une liste exhaustive des tâches de bureau.

3.4.2 La classe objet informationnel (O.I.)

La classe objet informationnel représente l'information se trouvant dans le bureau et utilisée lors de l'exécution d'une tâche. Cette information se trouve inscrite au moyen du langage naturel ou d'un autre type de langage sur un support physique que l'on peut manipuler; ce support est généralement le papier.

> La classe objet informationnel est l'ensemble des objets du bureau qui sont constitués d'un support et de l'information qui s'y trouve inscrite.

Dans un bureau qui traite de l'information, le nombre d'objet appartenant à la classe objets informationnels est très important. Parmi tous ces objets cependant, on peut en distinguer un certain nombre que l'on ne peut pas décomposer; si on prend pour exemple une feuille de papier, sans la couper, on ne peut pas en obtenir deux. Ce seront les objets informationnels élémentaires.

D'un autre côté, on peut regrouper plusieurs objets informationnels élémentaires dans un seul objet. Si on décompose cet objet, on obtiendra les objets informationnels élémentaires qui le constituaient. Cet objet composé fera partie de la classe des objets informationnels structurants.

3.4.2.1 La classe objet informationnel élémentaire (O.I.E.)

De ce que l'on vient de dire, nous pouvons en arriver à la définition de la classe objet informationnel élémentaire.

> La classe d'O.I.E. est l'ensemble des objets qui sont constitués d'un support et de l'information qui s'y trouve inscrite. Cependant, le support ne peut pas, logiquement, être décomposé pour donner d'autres objets.

Sur une feuille de papier, il y a cependant plusieurs façons d'y inscrire l'information; on peut l'écrire en vrac sans soins particuliers, on peut l'écrire dans certaines parties de la feuille à des endroits fixés et libellés par un titre, on peut la représenter sous forme de graphiques, de tableaux, ...

Parmi l'ensemble des O.I.E., on s'aperçoit que l'information qui s'y trouve respecte certaines structures de mise en page; la structure de mise en pages utilisée est celle qui est la plus appropriée au texte à écrire. Il est donc possible de regrouper en classes tous les O.I.E. présentant les mêmes caractéristiques au point de vue de la manière d'inscrire l'information sur le support. Ces nouvelles classes vont constituer des sous-classes de la classe O.I.E. et sont au nombre de trois (cette classification est reprise de [DACH 86]).

a) La classe message

La classe message est l'ensemble des messages que l'on trouve dans le bureau. C'est un support d'information qui n'est pas décomposable, destiné à recevoir une information brève qui y sera écrite sans soins particuliers au point de vue du plan (l'information est trop brève) et de la mise en pages.

Le message est généralement utilisé pour communiquer l'une ou l'autre information à d'autres personnes ou pour servir d'aide mémoire. En général, il n'est pas destiné à sortir du cadre de l'organisation où se trouve le bureau.

b) La classe formulaire

La classe formulaire est l'ensemble des formulaires que l'on trouve dans le bureau. C'est un support d'information qui n'est pas décomposable, constitué d'un certain nombre de champs libellés par un titre et pouvant recevoir une valeur; pour inscrire une information sur un formulaire, on le fait en remplissant ces champs.

Le formulaire peut être considéré comme un questionnaire que l'on demande à quelqu'un de remplir et qu'il devra, dans la plupart des cas, réexpédier à celui qui le lui a donné ou envoyé.

c) La classe document

La classe document est l'ensemble des documents que l'on trouve dans un bureau. C'est un support d'information qui n'est pas décomposable, destiné à recevoir une information plus volumineuse et dont la mise en pages doit être soignée; les graphiques, les tableaux et les autres formes de présentation de l'information peuvent être utilisés.

Le document est généralement utilisé pour des comptes-rendus de réunions, des notes explicatives, des rapports divers, des lettres envoyées aux clients, ...

3.4.2.2 La classe objet informationnel structurant (O.I.S.)

La classe O.I.S. est une sous-classe de la classe objets informationnels et on peut la définir comme suit :

> La classe objet informationnel structurant est l'ensemble des objets qui sont composés par des O.I.E. et/ou par d'autres O.I.S. Un O.I.S. peut donc être décomposé; on peut à tout moment lui retirer un de ses constituants. Un O.I.S. est un support d'information en ce sens qu'il est composé par d'autres supports d'information. L'information qu'il représente équivaut à la réunion de toutes les informations inscrites sur les supports qui le composent.

Cependant parmi tous les regroupements possibles d'objet informationnel, on constate que certains d'entre eux possèdent des propriétés communes. Dans ces propriétés, on pourrait citer par exemple la classe des objets qui sont regroupés, les ordres possibles de classement, la raison du regroupement, ... On peut alors regrouper les O.I.S. qui présentent les mêmes caractéristiques pour former de nouvelles classes (cette classification est reprise de [DACH 86]).

a) La classe dossier

> La classe dossier est l'ensemble des dossiers du bureau. Ceux-ci sont constitués d'une part, d'un regroupement d'objets informationnels et d'autre part, de l'information se trouvant inscrite sur chacun des objets informationnels regroupés. En général, les informations regroupées sont relatives à un même sujet; par conséquent, la classe des objets se trouvant dans un dossier n'a pas d'importance.

Par exemple, on peut considérer le dossier d'un client douteux qui peut être composé de pièces très différentes telles que des factures impayées, des rappels à l'ordre ou encore ses coordonnées sous forme d'un formulaire. Toutes ces pièces sont relatives aux créances douteuses d'un client.

Nous devons cependant introduire une restriction; comme nous le verrons ci-dessous, une pile n'a absolument, par définition du dossier et de la pile, aucune raison de se trouver dans un dossier. On ne pourra donc pas trouver de pile dans un dossier.

b) La classe fichier

> La classe fichier est l'ensemble des fichiers d'un bureau. Ceux-ci sont constitués d'une part, d'un regroupement d'objets informationnels et d'autre part, de l'information se trouvant inscrite sur chacun des objets informationnels regroupés. La particularité d'un fichier est qu'il est composé d'objets appartenants à la même classe.

Par exemple, un fichier fournisseur est composé d'un ensemble de formulaires; chacun d'eux reprenant les renseignements signalétiques d'un fournisseur. Les formulaires peuvent être classés selon un certain ordre : triés sur le nom du fournisseur par exemple.

Pour les mêmes raisons que pour le dossier, on ne peut pas trouver de pile dans un fichier.

Comme nous le verrons plus tard, un document, un formulaire ou un dossier peut avoir un type. Le type peut être considéré comme une classification à l'intérieur de la classe formulaire par exemple. Comme c'est l'utilisateur qui décide des types qu'il veut créer, il nous est impossible de les déterminer et donc de créer les sous-classes adéquates dans les classes formulaire, document et dossier.

Nous avons introduit la notion de type dès maintenant car si les constituants d'un fichier sont des formulaires, des documents ou des dossiers, ils doivent tous être non seulement de la même classe mais aussi du même type.

c) La classe pile

La classe pile est l'ensemble des piles d'un bureau. Celles-ci sont constituées d'une part, d'un regroupement d'objets informationnels et d'autre part, de l'information se trouvant inscrite sur chacun des objets informationnels regroupés. Dans une pile, on peut y trouver n'importe quel objet (sauf d'autres piles) qui n'ont aucuns liens logiques entre eux et qui sont classés dans l'ordre chronologique inverse; mais cet ordre n'est pas intentionnel.

Dans un bureau, le courrier reçu est souvent mis sur une pile car en général tout ce qui est reçu est empilé sur le tas du courrier déjà existant et en attente de traitement.

Avec cette définition de la pile, on comprend aisément que l'on ne peut pas trouver de pile dans un fichier ou dans un dossier; ceux-ci exigent en effet qu'il y ait une certaine relation entre les objets regroupés.

d) L'ordre de classement dans la classe O.I.S.

Un O.I.S. étant un regroupement d'objets informationnels, nous allons regarder dans quel ordre ceux-ci vont pouvoir être classés dans l'O.I.S. Si on reprend l'exemple du dossier du client douteux, dans quel ordre va-t-on classer les pièces du dossier ? Pour le fichier signalétique des fournisseurs, comment va-t-on ordonner les fiches relatives aux fournisseurs ?

L'ordre de classement associé à un O.I.S. définit la place, parmi tous les objets informationnels déjà classés, à laquelle sera placé un nouvel objet informationnel. Il permettra aussi de trouver la place d'un objet que l'on recherche.

De plus, l'ordre de classement associé à un O.I.S. permet de spécifier l'ordre d'apparition des objets informationnels qui y sont classés, lorsqu'on le feuillète.

Le nombre d'ordre de classement sera limité puisque nous ne gérons pas le système d'information propre aux objets et nous n'en retiendrons dès lors que cinq :

- l'ordre LIFO,
- l'ordre FIFO,
- l'ordre ALPHABETIQUE,
- l'ordre ALPHABETIQUE INVERSE,
- aucun ordre.

Nous définissons ci-dessous les différents ordres de classement possibles; pour chacun d'eux, on décrira comment sera inséré un objet dans un O.I.S. et ensuite l'ordre d'apparition des objets lorsqu'on feuillète l'O.I.S. :

- FIFO
  - l'objet informationnel inséré se trouve placé derrière tout ceux qui s'y trouvent déjà.
  - lorsqu'on feuillète l'O.I.S., c'est le premier objet classé qui apparait en premier.

- LIFO
  - l'objet informationnel inséré se trouve placé devant tout ceux qui s'y trouvent déjà.
  - lorsqu'on feuillète l'O.I.S., c'est le dernier objet classé qui apparait en premier.

- ALPHABETIQUE
  - l'objet informationnel inséré se trouve placé derrière celui dont le nom lui est directement inférieur dans l'ordre alphabétique.
  - lorsqu'on feuillète l'O.I.S., c'est l'objet dont le nom est le premier dans la liste alphabétique de tous les noms des autres objets qui apparait en premier.

- ALPHABETIQUE INVERSE
  - l'objet informationnel inséré se trouve placé derrière celui dont le nom lui est directement supérieur dans l'ordre alphabétique.
  - lorsqu'on feuillète l'O.I.S., c'est l'objet dont le nom est le dernier

dans la liste alphabétique de tous les noms des autres objets qui apparait en premier.

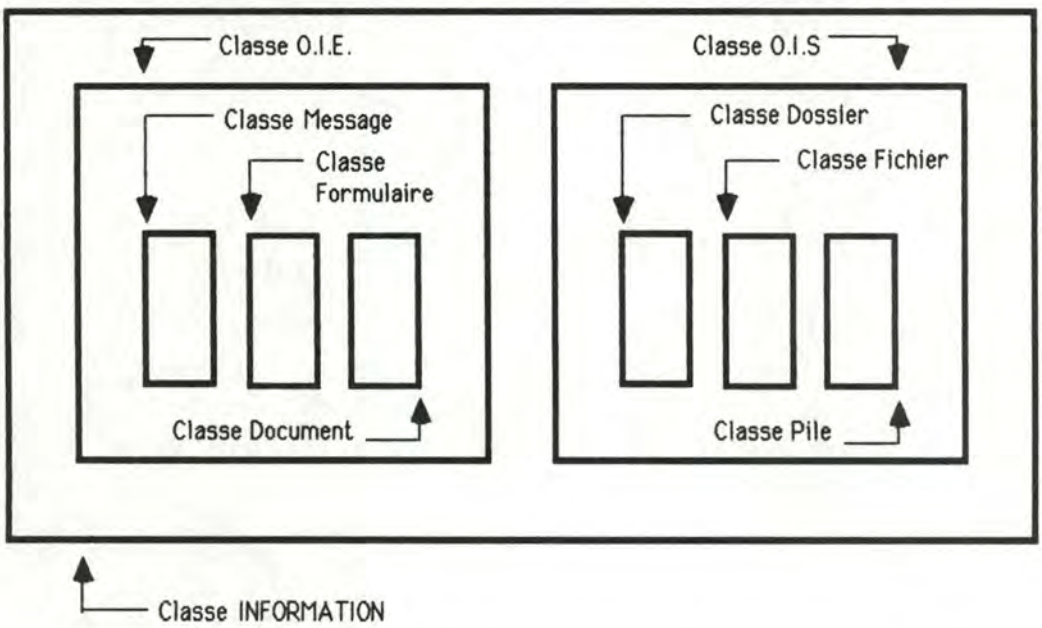
- AUCUN
  - l'objet informationnel inséré se trouve placé n'importe où dans l'O.I.S.
  - lorsqu'on feuillète l'O.I.S., les objets apparaissent de façon quelconque.

Remarques :

- ces ordres peuvent être appliqués à tous les types d'O.I.S. sauf pour la classe pile où seul l'ordre LIFO sera géré.
- l'ordre de classement d'un O.I.S. est spécifié lors de sa création.
- lorsqu'on désire feuilletter un O.I.S., on peut désirer accéder aux objets qui s'y trouvent dans un ordre différent. Ainsi, sans changer l'ordre initial, on accède aux informations selon un ordre différent.
- pour tout O.I.S., il est possible, à tout moment, de modifier son ordre initial en le triant selon un autre ordre de classement.

e) Résumé de la super-classe objet informationnel

La décomposition des objets informationnels est résumée à la figure 3.5.



La classe INFORMATION est la super-classe des classes O.I.E. et O.I.S.
La classe O.I.E. est la super-classe des classes message, formulaire et document
La classe O.I.S. est la super-classe des classes dossier,fichier et pile.
Les classes message, formulaire et document sont des sous-classes de

la classe O.I.E.
Les classes dossier, fichier et pile sont des sous-classes de la classe O.I.S.
Les classes O.I.S. et O.I.E. sont des sous-classes de la classe INFORMATION

Fig. 3.5 : Structuration de la super-classe INFORMATION

3.4.3 La classe objet de rangement

Dans un bureau, tout objet informationnel possède un endroit défini où il se trouve rangé. Pour définir ces différents lieux de rangement, nous parlerons d'objets de rangement.

> La classe objet de rangement caractérise l'ensemble des objets de rangement du bureau. Ces objets doivent servir de lieu de rangement à des objets informationnels.

Un objet de rangement peut être composé par d'autres objets : par exemple, une armoire est composée de tiroirs. N'importe quel objet ne peut toutefois se trouver n'importe où : une armoire ne peut être contenue dans une farde ! Dans l'ensemble des objets de rangement, nous pouvons regrouper tout ceux qui matériellement possèdent les mêmes caractéristiques. La caractéristique essentielle d'un objet de rangement est sa contenance; que peut-on y trouver ? Des O.I.E., des O.I.S. ou d'autres objets de rangement ... Si oui, de quels types sont ces objets ? Cela nous amène à créer des sous-classes de la classe objet de rangement (cette classification est établie à partir de [ROBE 86]).

a) La classe farde

> La classe farde est l'ensemble des fardes du bureau. Celles-ci ne sont composées par aucun autre objet de rangement et on peut y ranger n'importe quel objet informationnel.

La farde peut avoir plusieurs aspects : elle peut être en plastique, en carton, avoir ou non des anneaux, ...

b) La classe tiroir

> La classe tiroir est l'ensemble des tiroirs du bureau. Ceux-ci peuvent contenir des fardes et on peut y ranger n'importe quel objet informationnel.

c) La classe armoire

La classe armoire est l'ensemble des armoires du bureau. Celles-ci peuvent contenir des tiroirs. Pour ranger de l'information dans une armoire, on doit le faire via ses tiroirs; on ne peut pas ranger un document dans une armoire sans qu'il ne soit dans un tiroir.

d) La classe boîte

La classe boîte regroupe l'ensemble des boîtes du bureau. Celles-ci peuvent contenir des fardes et on peut y ranger n'importe quel objet informationnel.

Une boîte est en général en carton et sert dans la plupart des cas à archiver l'information de bureau qui n'est plus pertinente.

e) La classe étagère

La classe étagère est l'ensemble des étagères du bureau. Celles-ci peuvent contenir des boîtes et des fardes et on peut y ranger n'importe quel objet informationnel.

f) Résumé de la structuration des objets de rangement

La structuration des objets de rangement est résumée à la figure 3.6.

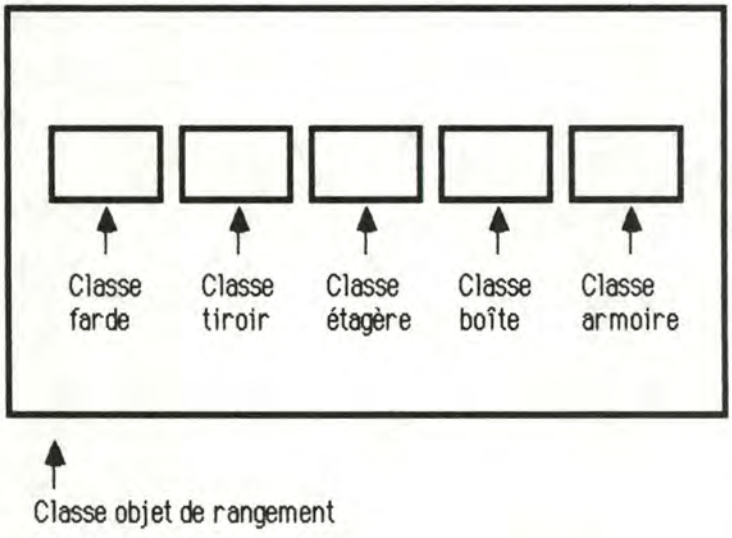


Fig. 3.6 : La structure des objets de rangements

3.4.4 La classe outil de travail

Dans un bureau, il existe généralement de la technologie qui facilite l'exécution d'une tâche. Cette technologie englobe déjà la classe objet de

rangement vue au point précédent. En plus de celle-ci, il faut inclure les objets qui permettent à l'utilisateur de traiter de l'information matérialisée par la classe objet informationnel. Ces objets seront regroupés dans la classe outil de travail. Nous en donnons la définition suivante :

> La classe outil de travail regroupe les objets qui permettent à la personne qui exécute la tâche, de travailler sur l'information dont elle dispose.

Parmi les objets de cette classe, on peut distinguer des objets qui ont des fonctions tout à fait différentes. La façon dont ces objets sont utilisés par l'utilisateur va nous permettre de regrouper ces objets en classes.

a) La classe table de travail

Comme nous l'avons décrit lors de la description du diamant de Leavitt, une tâche est constituée d'un ensemble d'opérations primitives que la personne exécute sur l'information rangée dans le bureau. La table de travail va permettre à la personne de traiter cette information; lire un document, consulter un formulaire, ... L'information utilisée par la personne est amenée sur la table de travail et y reste jusqu'à ce qu'elle soit replacée à sa place ou rangée à un autre endroit.

> La classe table de travail est l'ensemble des tables de travail du bureau. Elles permettent à l'utilisateur de travailler sur l'information qu'il utilise pour réaliser sa tâche.

A la table de travail, nous ne donnons pas d'autre fonction que celle du traitement. Ainsi, tout objet informationnel qui se trouve sur ou dans la table de travail, ne s'y trouve que temporairement et n'y est donc pas rangé. La table de travail peut avoir des tiroirs mais ceux-ci ne sont pas considéré sur le même pied que les tiroirs qui se trouvent dans une armoire.

C'est essentiellement dans un but de clareté que nous apportons cette limitation. Il est en effet très difficile de parler de la table de travail à la fois comme outil de travail et à la fois comme objet de rangement sans entraîner des confusions.

b) La classe photocopieuse

La photocopieuse a certainement un rôle moins important que la table de travail car elle ne servira que lorsque la personne veut dupliquer un objet informationnel. Tout comme pour la table de travail, l'information qui est utilisée reste sur la photocopieuse jusqu'à ce qu'elle soit replacée à sa place ou rangée à un autre endroit. Quant aux copies, elles resteront également sur la photocopieuse jusqu'à ce que la personne les range.

La classe photocopieuse est l'ensemble des photocopieuses du bureau. Elles permettent à la personne qui exécute une tâche de réaliser des copies d'objets informationnels élémentaires.

Dans le cadre de ce mémoire, nous nous fixons comme convention de représenter l'environnement du bureau qui est propre à une seule personne. Cette convention n'est nullement limitatrice pour la suite du projet dans lequel s'insère ce mémoire. On peut en effet considérer que lorsque plusieurs personnes travaillent ensemble, elles exécutent chacune une tâche. La communication inter-tâche est donc une possibilité pour simuler plusieurs personnes qui travaillent ensemble. (la communication inter-tâche sera de nouveau évoquée dans le chapitre 4 au point 4.5.1).

Dans le contexte qui vient d'être décrit, nous pouvons dire que dans la classe table de travail et dans la classe photocopieuse, un seul objet sera suffisant.

3.4.5 La classe objet d'interface

D'après ce qui a été dit jusqu'à présent, on peut considérer le bureau comme une unité autonome dans laquelle une personne exécute des tâches. Cette unité ne peut cependant pas fonctionner si elle ne peut soit communiquer des objets informationnels vers l'extérieur du bureau soit recevoir des objets informationnels de cet extérieur. L'extérieur du bureau peut être soit un autre bureau de la même organisation soit un bureau d'une autre organisation ou soit un endroit qui ne s'apparente pas avec la définition d'un bureau.

Les objets qui permettent de réaliser ces transferts peuvent être considérés comme de la technologie (cfr. la figure 3.2) puisqu'ils permettent à la personne d'exécuter sa tâche. Ces objets seront regroupés dans la classe objet interface dont nous donnons la définition suivante :

> La classe objet interface regroupe les objets du bureau qui permettent soit de communiquer de l'information vers l'extérieur du bureau soit de recevoir de l'information en provenance de cet environnement. Par cet aspect des choses, nous dirons que tout objet informationnel qui se trouve dans un objet d'interface, ne s'y trouvera que temporairement.

Parmi tous les objets qui forment la classe objet interface, nous pouvons constater que certains d'entre eux ont des propriétés spécifiques. Dans ces propriétés nous trouvons la classe des objets qui sont échangés, la fonction des objets d'interface, le moyen de communication utilisé, ... Il serait dès lors utile de créer à l'intérieur de la classe objet interface, de nouvelles classes qui regrouperont les objets répondant aux mêmes propriétés.

a) La classe téléphone

La classe téléphone regroupe les objets qui permettent de transmettre l'information au moyen de la voix. Cette transmission se faisant dans le sens de la communication et de la réception.

b) La classe poubelle

La classe poubelle regroupe les objets permettant de détruire tout objet informationnel. Cette information transite du bureau vers la décharge publique et uniquement dans ce sens.

c) La classe boîte IN

La classe boîte IN regroupe les objets permettant de recevoir tout objet informationnel en provenance de l'environnement du bureau.

d) La classe boîte OUT

La classe boîte OUT regroupe les objets permettant de communiquer des objets informationnels vers l'environnement du bureau.

En ce qui concerne les objets de ces nouvelles classes, on peut faire la même remarque faite à propos des outils de travail. Puisque nous considérons le lieu de travail d'une seule personne, il n'y aura au maximum qu'un seul objet dans ces nouvelles classes.

3.5 Modèle de représentation

Cette analyse du bureau ayant été réalisée selon l'optique des langages orientés objets, il faut maintenant pouvoir en donner une représentation formelle avec un modèle qui permette de représenter les objets que nous avons décrits (rôle conceptuel). Ce modèle doit de plus être structuré pour permettre une translation directe du schéma conceptuel dans des structures de données physiques définies par un SGBD (rôle représentatif). [ELMA 85]

Le modèle que nous allons utiliser est le modèle Entité-Association présenté par Chen [CHEN 76] et que nous l'avons choisi par pure convenance; nous avons cependant été influencé par des avis tel que celui donné dans [BOPI 83] :

"l'audience croissante qu'il rencontre parmi les spécialistes des systèmes d'information et des bases de données. Un large consensus semble se dégager pour reconnaître aux modèles relevant de cette approche de grandes qualités de communication et une bonne capacité de représentation opérationnelle des informations appartenant au réel perçu".

3.6 Correspondance entre la décomposition en objets et le modèle E/A

Nous allons établir la correspondance entre les concepts développés dans l'introduction sur les langages orientés objets et les concepts du modèle Entité-Association (E/A). Lorsque celle-ci sera réalisée, nous pourrons exprimer le modèle du bureau avec les concepts du modèle E/A.

3.6.1 Représentation des concepts du langage objet

3.6.1.1 La représentation d'une classe

La première translation que l'on peut faire va se réaliser entre le concept de classe et celui de type d'entité. Donnons la définition de chacun de ces concepts et comparons-les :

- une classe est une description d'un ou plusieurs objets similaires qui représentent un même aspect du réel [ROBS 81].
- un type d'entité est la classe de toutes les entités possibles du réel perçu qui vérifient la définition constitutive du type. Les types d'entité ne forment pas nécessairement des classes disjointes [BOPI 83].

Une classe ou un type d'entité définit donc les propriétés communes de tous leurs objets dans le cas d'une classe et de toutes leurs entités dans le cas d'un type d'entité. Ces objets ou ces entités représentant un même aspect du réel. Ainsi, nous pouvons assimiler une classe à un type d'entité.

3.6.1.2 La représentation d'une sous-classe et d'une super-classe

Pour la représentation des concepts de sous-classe et de super-classe, une difficulté va cependant se présenter car dans le modèle Entité-Association rien n'est prévu pour les représenter. Dans un article récent [ELMA 85] cette difficulté est clairement énoncée :

> "des recherches continues dans les modèles de données semblent indiquer que les concepts du modèle de données E/A ne sont pas suffisants pour représenter certains aspects sémantiques importants. Les concepts de sous-classe et de super-classe ne sont pas directement supportés par le modèle Entité-Association".

Pour remédier à ce problème, on voit apparaître des modèles Entité-Association étendus qui sont des extensions possibles au modèle de base. Parmi ces modèles étendus, citons :

- ECR : Entity-Category-Relationship exposé dans [ELMA 85] propose la notion de catégorie qui peut représenter les concepts de sous-classe et

de super-classe. La catégorie y est présentée comme étant un ensemble d'entités pouvant appartenir à un ou plusieurs types d'entités. Une catégorie peut participer à une association avec une autre catégorie ou avec un type d'entité. La notion de super-classe est représentée par une catégorie qui englobe plusieurs entités de types différents. La notion de sous-classe est représentée par une catégorie qui est un sous-ensemble des entités d'un même type.

- Dans l'article [LINE 86], les notions de sous-classe et de super-classe sont représentées par un type d'entité. Ainsi, un type d'entité E est appelé super-classe d'un type d'entité E' s'il y a une association d'inclusion entre les deux : E' inclut ou égal à E. Cela signifie que chaque occurrence de type E' appartient aussi au type E. On dira que E' est une sous-classe de E. De plus, E' hérite de tous les attributs de E de façon implicite. La représentation graphique est donnée à la figure 3.7.

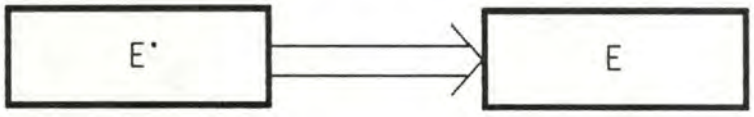


Fig. 3.7 : E' est une sous-classe de E

Pour la représentation des concepts de sous-classe et de super-classe, nous allons utiliser la notion de sous-typage qui n'est pas définie dans le modèle E/A de base. Elle représente néanmoins une notion importante au sein des modèles de données. La littérature à ce sujet laisse beaucoup de libertés quant à sa définition et à son utilisation. C'est pourquoi, nous définirons notre propre concept de sous-type. Pour la définition, nous nous référerons à la figure 3.8.

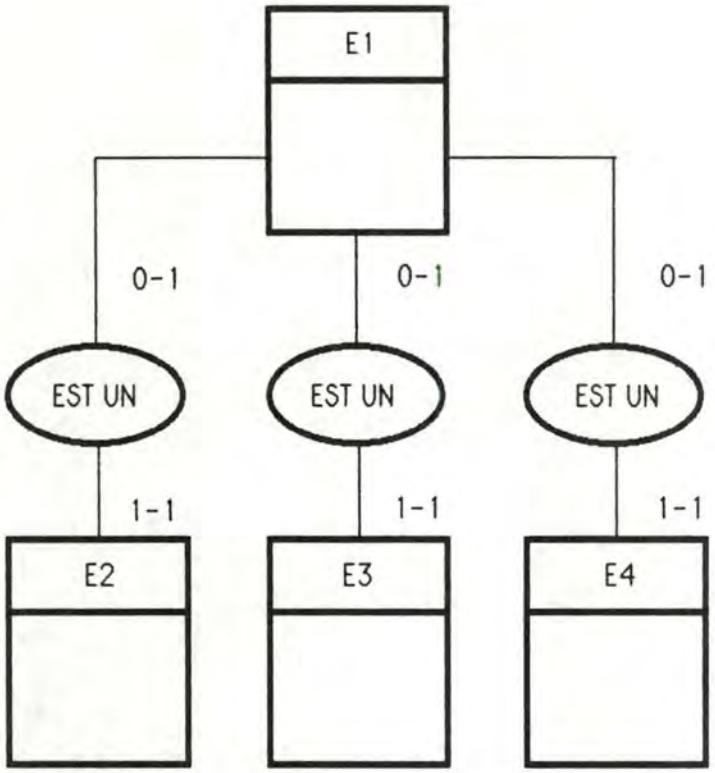


Fig. 3.8 : Représentation d'une relation de sous-typage

Nous appelerons E2 sous-type de E1 si et seulement si toutes les occurrences de E2 appartiennent aussi à E1. (la définition est aussi valable pour E3 et E4).

Sur le schéma de la figure 3.8, la notion d'appartenance est représentée par une relation *est un*. Une autre notation très souvent utilisée est *inclut ou égal*.

Suite à cette définition, plusieurs possibilités sont possibles :

- l'ensemble des occurrences de E2, de E3 et de E4 constituent l'ensemble des occurrences de E1,
- l'ensemble des occurrences de E2, de E3 et de E4 constituent un sous-ensemble des occurrences de E1,
- l'intersection entre les sous-types peut être vide ou non,
- les occurrences d'un sous-type peuvent appartenir à plus d'un type d'entité,
- les occurrences de E2, de E3 et de E4 héritent implicitement des attributs de E1.

3.6.1.3 Représentation de l'information propre à un objet

L'information propre à un objet sera représentée dans le modèle E/A par le concept d'attribut. Nous en donnons la définition extraite de [BOPI 83] :

"Un attribut est une caractéristique ou une qualité d'une entité ou d'une association. Elle peut prendre une ou plusieurs valeurs ou groupe de valeurs."

3.6.1.4 Représentation d'un objet

D'après les propriétés des langages orientés objets, on peut dire que les relations entre les super-classe et les sous-classes vont représenter une hiérarchie (dans le cas de SMALLTALK-80, nous aurions une arborescence). Aussi lorsque ces concepts seront traduits dans le modèle E/A, nous aurons également une hiérarchie (cfr. les figures 3.9 b et c).

Pour représenter un objet qui se trouve dans une classe n, on le représente par une occurrence dans le type d'entité qui se trouve au sommet de la hiérarchie (à la figure 3.9 c, c'est le type d'entité OBJET). Dans ce type d'entité, on trouvera les informations communes à tous les objets du système de même que l'identification de l'objet. L'identification de l'objet correspond au nom qu'il possède dans le langage objet.

En plus, pour chaque classe dans laquelle l'objet se trouve soit directement soit via ses super-classes, on trouvera dans le type d'entité représentant la classe, une occurrence représentant l'information qui est propre à l'objet dans cette classe. Il faut remarquer que l'identification de l'objet ne se trouve pas reprise dans ces occurrences.

Par exemple, l'objet indiqué à la figure 3.9 c sera représenté par une occurrence dans le type d'entité OBJET, une occurrence dans le type d'entité CLASSE 3 et ... une occurrence dans le type d'entité CLASSE n; toute ces occurrences étant reliées par des relations *est un*.

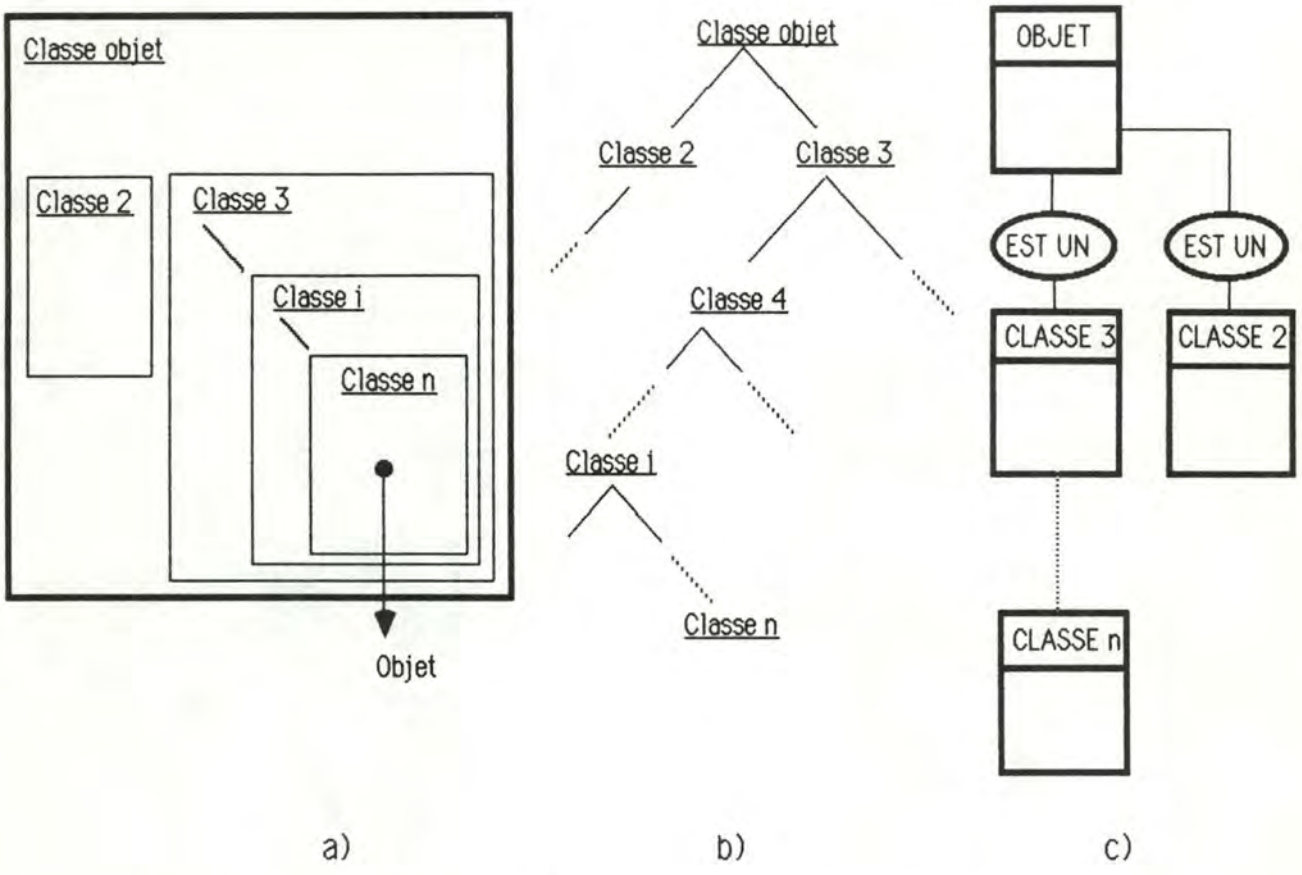


Fig. 3.9 : a) Représentation de classes b) Hiérarchie de classes c) Représentation des classes en types d'entités

3.6.1.5 Correspondance des propriétés

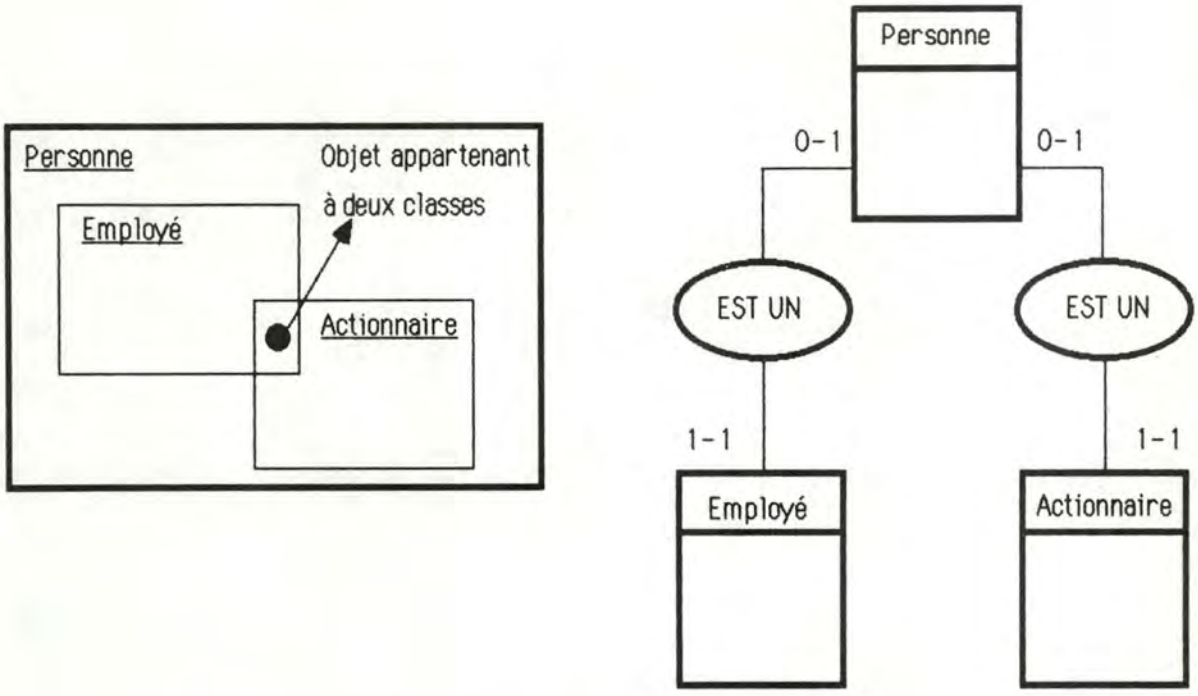
Jusqu'à présent, nous en sommes arrivés aux correspondances suivantes :

- classe --> type d'entité.
- objet --> une ou plusieurs occurrences.
- sous-classe --> sous-type.
- super-classe --> implicite dans la notion de sous-type.
- information --> attribut.

Voyons maintenant comment s'énonce les propriétés propres aux langages orientés objets décrites lors de la présentation du langage SMALLTALK-80, nous pouvons faire la correspondance suivante :

a) Un objet peut appartenir à une ou plusieurs classes

La transformation de cette contrainte peut s'expliquer à partir d'un exemple donné à la figure 3.10



a) b)

Fig. 3.10 a) Un objet appartient à deux sous-classes b) Représentation dans E/A

Sur le schéma de la figure 3.10 b., à toute personne peut lui être associée zéro, une ou deux occurrences de sous-types suivant que la personne n'est ni un employé ni un actionnaire, que la personne soit un employé ou un actionnaire , que la personne soit un employé et un actionnaire.

La notion d'un objet qui appartient à plusieurs sous-classes d'une même classe est représentée dans le modèle E/A par le fait que toute occurrence du type d'entité qui représente la super-classe peut être associée à des occurrences appartenant à des sous-types différents.

b) Toute sous-classe appartient à une super-classe

Un sous-type est toujours associé à un type d'entité.

c) Une classe peut avoir plusieurs sous-classes

Un type d'entité peut avoir plusieurs sous-types.

d) Une classe peut être sous-classe d'une ou de plusieurs autres classes

Un type d'entité peut être sous-type d'un ou de plusieurs autres types d'entité.

e) Une super-classe et une sous-classe sont aussi des classes

Un sous-type est aussi un type d'entité.

f) Une classe est un objet

Un type d'entité pourrait être une occurrence d'un type d'entité qui décrit le schéma Entité-Association sous la forme d'un méta-schéma.

g) Le principe d'héritage

Un sous-type hérite des attributs et des associations du type d'entité auquel il est associé. Pour retrouver l'information propre à un objet, il suffit à partir de l'occurrence du type d'entité OBJET représentant l'objet de descendre, par les relations *est un,* dans la hiérarchie et de prendre l'information se trouvant dans les occurrences associées directement ou non à l'occurrence de départ.

3.6.2 Représentation des objets de bureau dans le modèle E/A

Nous disposons à ce stade :

- de l'analyse du bureau dans l'optique des langages orientés objets,
- d'un modèle de représentation qui est le modèle Entité-Association,
- de la représentation des concepts des langages orientés objets dans le modèle E/A.

Il reste à réaliser effectivement la représentation des objets de bureau avec les concepts du modèle E/A.

a) La super-classe INFORMATION

En appliquant le principe de translation, nous obtenons le schéma de la figure 3.11.

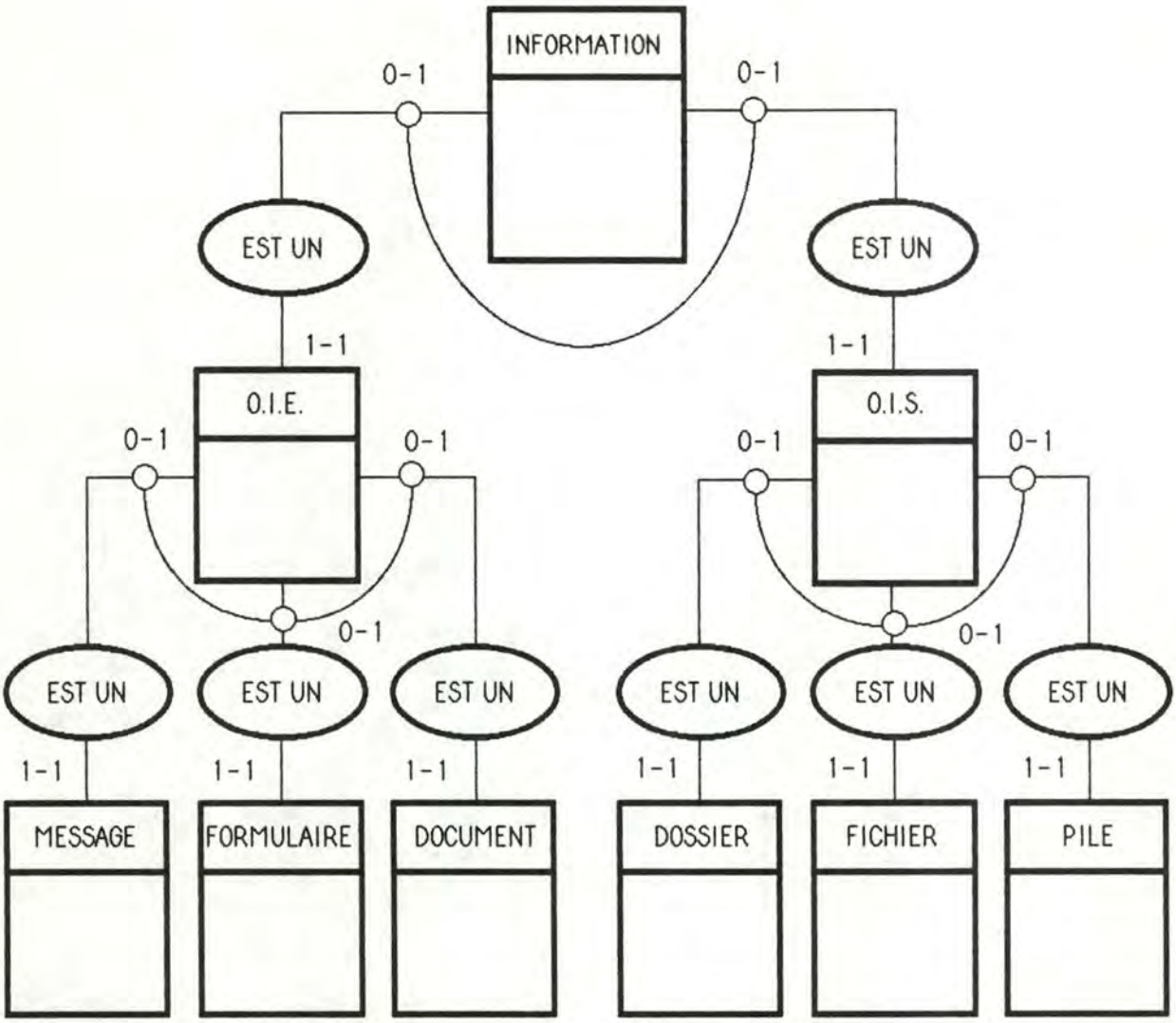


Fig. 3.11 : Représentation de la super-classe INFORMATION

Sur la figure 3.11, la super-classe INFORMATION y est représentée par un type d'entité et ses deux sous-classe O.I.E. et O.I.S. lui sont associées en tant que sous-types. Il existe cependant une relation d'exclusion entre les deux puisque un objet informationnel est soit un O.I.S. soit un O.I.E. : cela signifie que l'intersection entre les deux sous-classes est vide.

Les mêmes constations peuvent être tenues pour la classe O.I.E. qui est représentée par un type d'entité et ses trois sous-classes MESSAGE, FORMULAIRE et DOCUMENT lui sont associées en tant que sous-types. Il existe cependant une relation d'exclusion mutuelle entre les trois puisque un O.I.E. est soit un message, soit un formulaire, soit un document : cela signifie que l'intersection entre les trois sous-classes est vide.

Enfin, la classe O.I.S. est représentée par un type d'entité et ses trois sous-classes DOSSIER, FICHIER et PILE lui sont associées en tant que sous-types. Il existe aussi une relation d'exclusion mutuelle entre les trois puisque un O.I.S. est soit un dossier, soit un fichier, soit une pile : cela signifie que l'intersection entre les trois sous-classes est vide.

b) La super-classe OBJET_RANGEMENT

La représentation de la super-classe OBJET_RANGEMENT est donnée à la figure 3.12

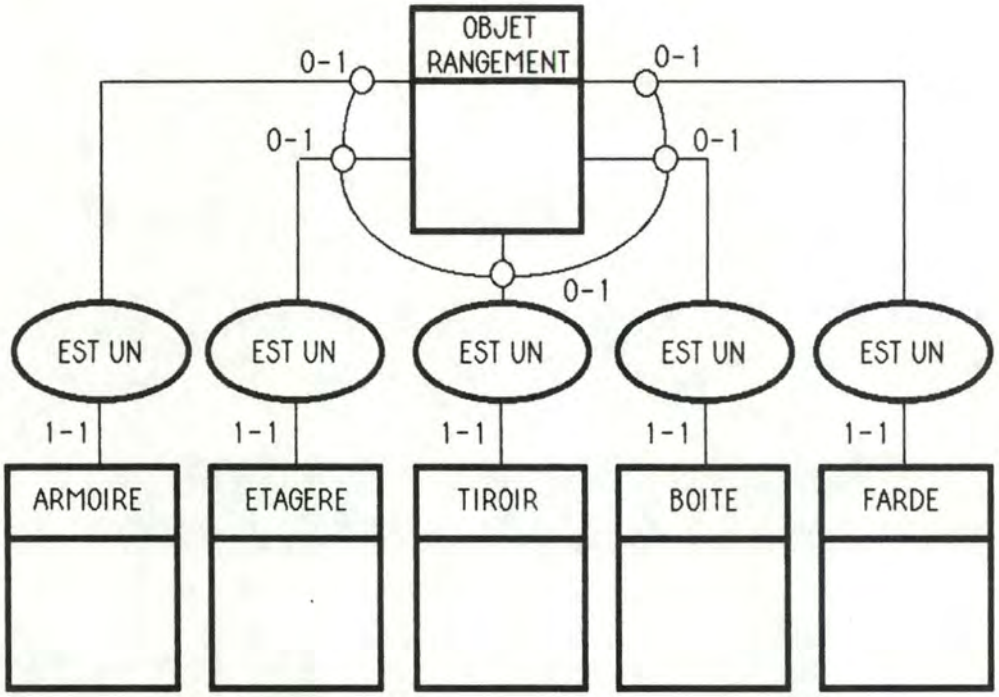


Fig 3.12 : La représentation de la super-classe OBJET_RANGEMENT

c) La super-classe OUTIL DE TRAVAIL

La représentation de la super-classe OUTIL_DE_TRAVAIL est donnée à la figure 3.13.

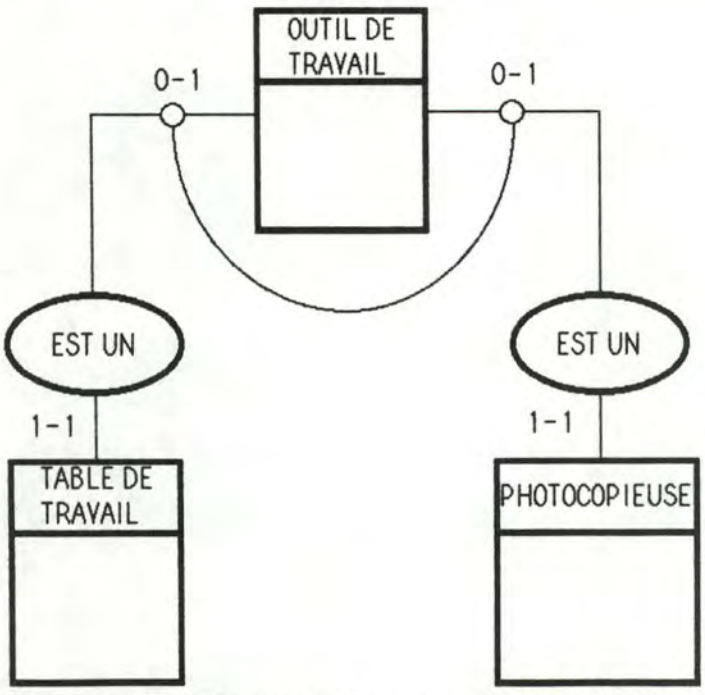


Fig. 3.13 : La représentation de la super-classe OUTIL_DE_TRAVAIL

d) La super-classe OBJET_INTERFACE

La représentation de la super-classe OBJET_INTERFACE est donnée à la figure 3.14.

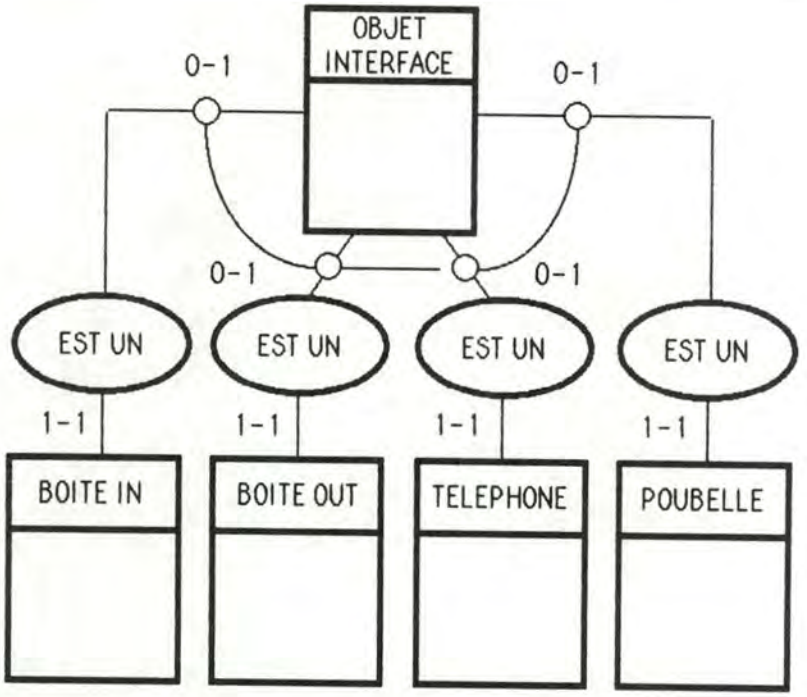


Fig. 3.14 : La représentation de la super-classe OBJET_INTERFACE

3.7 L'abstraction de données

3.7.1 Présentation du principe d'abstraction

Lorsque nous avons présenté les concepts de super-classe et de sous-classe, nous avions signalé que ce type de construction permettait d'introduire une abstraction sur les données.

Nous introduisons ci-dessous la notion d'abstraction et nous montrerons que dans notre travail, cette notion nous sera utile.

a) Définition

Cette définition est extraite de [SMSM 77] :

> "Une abstraction dans un système est un modèle de ce système dans lequel certains détails ont été délibérément omis. Le choix des détails qui sont omis est fait en considérant à la fois l'application qui va travailler sur le modèle et les utilisateurs de cette application."

L'objectif est de permettre de mettre en évidence certains détails du modèle qui sont indispensables à l'application et d'ignorer d'autres détails.

b) Avantages de travailler avec une hiérarchie d'abstractions

Lorsque dans le modèle de données on possède une hiérarchie, celle-ci permet d'introduire à tout moment des détails dans le modèle et cela de façon contrôlée. Travailler avec une hiérarchie permet aisément de réaliser une abstraction sur n'importe lequel de ses niveaux.

Une abstraction réalisée sur une hiérarchie de données introduit une grande stabilité pour les applications qui utilisent ces données. Une modification, comme une abstraction d'un niveau dans la hiérarchie, permet de ne pas modifier les niveaux supérieurs ; l'application peut donc continuer à travailler sur ces niveaux supérieurs en ignorant simplement le ou les niveaux supprimés.

c) La généralisation

Nous reprenons la définition de la généralisation donnée dans [SMSM 77] :

> "Une généralisation est une abstraction qui permet à un ensemble d'objets individuels d'être pensés et référencés par un concept général unique."

La généralisation est peut-être le mécanisme le plus important pour conceptualiser le monde réel. En effet, à tout moment, à partir de quelques cas isolés, nous avons toujours tendance à généraliser. Par exemple, si un enfant se fait mordre par un chien, il en déduira que tous les chiens mordent.

3.7.2 L'abstraction dans notre modèle de bureau

Bien que notre mémoire travaille sur la simulation graphique de tâches de bureau, il ne nous est pas nécessaire de travailler sur un modèle complet du bureau. Nous réalisons en effet un prototype et de ce fait, nous ferons abstraction de toute une série d'informations (de détails) qui ne vont pas nous concerner. Ces informations constituent le système d'information propres aux objets de bureau. Par exemple, le contenu d'un document ne nous intéresse pas; la seule chose que l'on veut savoir c'est qu'il s'agit d'un document.

Ceci nous amène à généraliser les objets informationnels, les objets d'interface, les objets de rangement et les outils de travail.

Il est très important de remarquer que dans les figures 3.11, 3.12, 3.13 et 3.14 nous avons des hiérarchies.

a) Généralisation des objets informationnels

Si on généralise les objets informationnels, représentés à la figure 3.11, cela nous amène à ne plus considérer que l'entité INFORMATION; on fait abstraction de tous les détails concernant les objets informationnels. Seule

l'information commune à tous les objets informationnels nous intéresse.

De manière à encore pouvoir connaître la classe des objets, nous ajoutons à l'entité INFORMATION l'attribut CLASSE.

La figure 3.15 représente la généralisation des objets informationnels.

Fig. 3.15 : Généralisation des objets informationnels

Sur la figure 3.15, les autres attributs font référence à l'information commune à tous les objets informationnels; quelle que soit leur classe. Nous décrirons en détail au point suivant, tous ces attributs.

b) Généralisation des objets de rangement

Si on généralise les objets de rangement, représentés à la figure 3.12, cela nous amène à ne plus considérer que l'entité OBJET RANGEMENT; on fait abstraction de tous les détails concernant les objets de rangement. Seule l'information commune à tous les objets de rangement nous intéresse.

De manière à encore pouvoir connaître la classe des objets, nous ajoutons à l'entité OBJET RANGEMENT l'attribut CLASSE.

La figure 3.16 représente la généralisation des objets de rangement.

Fig. 3.16 : Généralisation des objets de rangement

c) Généralisation des outils de travail

Si on généralise les outils de travail, représentés à la figure 3.13, cela nous amène à ne plus considérer que l'entité OUTIL DE TRAVAIL; on fait abstraction de tous les détails concernant les outils de travail. Seule l'information commune à tous les outils de travail nous intéresse.

De manière à encore pouvoir connaître la classe des objets, nous ajoutons à l'entité OUTIL DE TRAVAIL l'attribut CLASSE.

La figure 3.17 représente la généralisation des objets de rangement.

Fig. 3.17 : Généralisation des outils de travail

d) Généralisation des objets interface

Si on généralise les objets d'interface, représentés à la figure 3.14, cela va nous amène à ne plus considérer que l'entité OBJET INTERFACE; on fait abstraction de tous les détails concernant les objets d'interface. Seule l'information commune à tous les objets d'interface nous intéresse.

De manière à encore pouvoir connaître la classe des objets, nous ajoutons à l'entité OBJET INTERFACE l'attribut CLASSE.

La figure 3.18 représente la généralisation des objets de rangement.

Fig. 3.18 : Généralisation des objets d'interface

3.8 Schéma conceptuel du bureau

Nous disposons après avoir réalisé une abstraction sur des données non pertinentes de quatre types d'entité qui modélisent chacun un certain aspect du bureau. Cependant, nous ne disposons pas encore d'une représentation complète. Il manque en effet, les relations qui existent entre les types d'entité et qui expriment les faits suivants :

- une information peut être la copie d'une autre.
- une information peut être classée ou rangée.
- une information peut être en cours de traitement.
- un objet de rangement peut être composé d'autres objets de rangement.

Toutes ces relations vont être modélisées par des associations entre les types d'entité concernés. Nous obtenons donc le schéma de la figure 3.19.

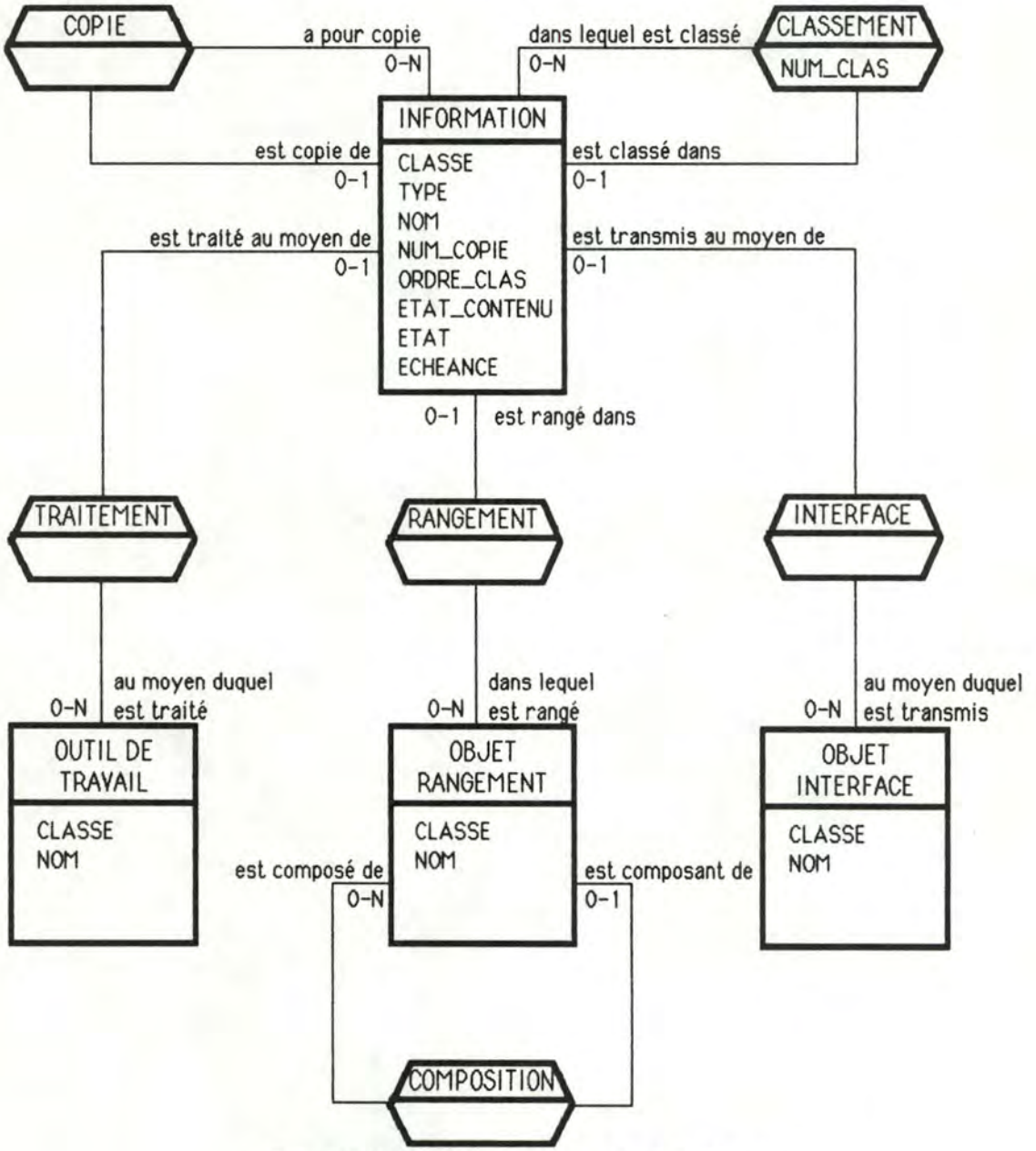


Fig. 3.19 : Schéma conceptuel du bureau.

Il nous reste désormais à définir les différents types d'entité et types d'association.

3.8.1 Description des types d'entité

Pour chaque type d'entité, nous donnerons sa définition, la liste de ses attributs et son identifiant. Pour chacun des attributs du type d'entité, nous donnerons sa définition, son type et enfin, son domaine de valeurs.

1) Le type d'entitié INFORMATION

a) Définition

Ce type d'entité regroupe les objets informationnels élémentaires et les objets informationnels structurants qui ont été décrits au point 3.4.2.

b) Attributs

Les attributs du type d'entité INFORMATION sont les suivants:

- CLASSE
  - cet attribut permet de connaître la classe d'une occurrence appartenant à ce type.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères suivant : [MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE].

- TYPE
  - cet attribut permet de répertorier en types les objets informationnels d'une même classe. Cet attribut n'est significatif et obligatoire que pour les formulaires, les documents et les dossiers. Cette sous-classification est introduite pour permettre à l'analyste de décrire des sous-classes de formulaires, de dossiers ou de documents. Avant la description de la tâche par l'analyste, le système ne connait pas les valeurs de cet attribut.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commencant par une lettre.

- NOM
  - cet attribut spécifie le nom de l'objet informationnel. La valeur de cet attribut est spécifiée par l'analyste lors de la description de l'environnement du bureau.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commencant par une lettre.

- NUM_COPIE
  - cet attribut spécifie le numéro de copie de l'objet informationnel. Cet attribut n'est significatif et obligatoire que pour les messages, les formulaires et les documents. Nous conviendrons qu'un numéro de copie nul indique qu'il s'agit d'un original.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui d'un nombre entier positif.

- ETAT
  - cet attribut est relatif au cycle de vie de l'objet informationnel. Il permet de connaître l'état d'avancement de l'objet informationnel dans son cycle de vie. La valeur de cet attribut est donnée par l'analyste lorsqu'il décrit sa tâche. Cette valeur pouvant changer en cours de tâche.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeur est l'ensemble des chaînes de caractères commencant par une lettre.

- ECHEANCE
  - cet attribut indique l'échéance à laquelle l'objet informationnel doit être traité au plus tard. La gestion en est laissée à l'utilisateur du système.
  - attribut de type élémentaire,facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères représentant une date sous le format AAAA/MM/JJ.

- ORDRE_CLAS
  - cet attribut spécifie l'ordre de classement dans lequel doivent se trouver les constituants de l'objet informationnel. Cet attribut n'est significatif et obligatoire que pour le dossier et le fichier. Pour la pile, l'ordre LIFO sera imposé automatiquement.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères suivant : [LIFO, FIFO, ALPHABETIQUE, ALPHABETIQUE INVERSE, AUCUN].

- ETAT_CONTENU
  - cet attribut indique si l'objet informationnel est vide. Cet attribut n'est significatif et obligatoire que pour le dossier, le fichier et la pile. Nous dirons qu'un objet informationnel est vide si physiquement, il ne possède aucun constituant. Cet attribut devra être géré automatiquement par le système.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères suivant : [VIDE, NON VIDE].

c) Identifiant

Une occurrence du type d'entité INFORMATION est identifiée par la combinaison de ses attributs CLASSE, TYPE, NOM et NUM_COPIE. Etant donné que certains de ces attributs sont facultatifs selon la classe de l'objet informationnel, le tableau 3.1 présente pour chacun la combinaison identifiante.

ATTRIBUTS / CLASSE	CLASSE	TYPE	NOM	NUM_COPIE
MESSAGE	X		X	X
FORMULAIRE	X	X	X	X
DOCUMENT	X	X	X	X
DOSSIER	X	X	X	
FICHIER	X		X	
PILE	X		X	

Tableau 3.1 : L'identifiant du type d'entité INFORMATION.

2) Le type d'entitié OBJET_RANGEMENT

a) Définition

Ce type d'entité regroupe les objets de rangement qui ont été décrits au point 3.4.3.

b) Attributs

Les attributs du type d'entité OBJET_RANGEMENT sont les suivants:

- CLASSE
  - cet attribut permet de connaître la classe d'une occurrence appartenant à ce type.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères suivant : [ARMOIRE, ETAGERE, TIROIR, BOITE, FARDE].

- NOM
  - cet attribut spécifie le nom de l'objet de rangement.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commencant par une lettre.

c) Identifiant

Une occurrence du type d'entité OBJET_RANGEMENT est identifiée par la combinaison de ses attributs CLASSE et NOM.

3) Le type d'entitié OUTIL_DE_TRAVAIL

a) Définition

Ce type d'entité regroupe les outils de travail qui ont été décrits au point 3.4.4.

b) Attributs

Les attributs du type d'entité OUTIL_DE_TRAVAIL sont les suivants:

- CLASSE
  - cet attribut permet de connaître la classe d'une occurrence appartenant à ce type.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères suivant : [TABLE_DE_TRAVAIL, PHOTOCOPIEUSE].

- NOM
  - cet attribut spécifie le nom de l'outil de travail.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commencant par une lettre.

c) Identifiant

Une occurrence du type d'entité OUTIL_DE_TRAVAIL est identifiée par la combinaison de ses attributs CLASSE et NOM.

4) Le type d'entitié OBJET_INTERFACE

a) Définition

Ce type d'entité regroupe les objets d'interface qui ont été décrits au point 3.4.5.

b) Attributs

Les attributs du type d'entité OBJET_INTERFACE sont les suivants:

- CLASSE
  - cet attribut permet de connaître la classe d'une occurrence appartenant à ce type.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères suivant : [BOITE IN, BOITE OUT, TELEPHONE, POUBELLE].

- NOM
  - cet attribut spécifie le nom de l'objet d'interface.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commencant par une lettre.

c) Identifiant

Une occurrence du type d'entité OBJET_INTERFACE est identifiée par la combinaison de ses attributs CLASSE et NOM.

3.8.2 Description des types d'association

Pour chaque type d'association, nous donnerons sa description, la liste des types d'entité qu'elle associe, la liste de ses attributs éventuels et les contraintes d'intégrité qui s'y rapportent. Pour chacun des types d'entité associés, nous donnerons le nom de rôle qu'il joue dans l'association et sa connectivité. Pour chacun des attributs du type d'association, nous donnerons sa définition, son type et enfin, son domaine de valeurs.

1) Le type d'association COPIE

a) Description

Le type d'association COPIE représente la duplication éventuelle d'un objet informationnel élémentaire.

b) Entités associées

Le type d'association COPIE associe de manière récursive le type d'entité INFORMATION.

Rôles : - "*est copie de*" pour INFORMATION (1).
- "*a pour copie*" pour INFORMATION (2).

Connectivités: - 0-1 pour INFORMATION (1)
- 0-N pour INFORMATION (2).

c) Contraintes d'intégrité

Seul des messages, des formulaires et des documents peuvent participer à une association COPIE.

Seul un original peut participer à une association COPIE par le rôle "*a pour copie*". Seul une copie peut participer à une association COPIE par le rôle "*est copie de*".

Si deux occurrences de INFORMATION sont associées par COPIE, alors elles doivent avoir des valeurs d'attribut identiques pour leurs attributs CLASSE, TYPE et NOM et une valeur d'attribut différente pour leur attribut NUM_COPIE.

2) Le type d'association CLASSEMENT

a) Description

Le type d'association CLASSEMENT représente le classement d'un objet informationnel élémentaire ou structurant dans un objet informationnel structurant. Si un objet informationnel classé participe à une association TRAITEMENT, alors il est physiquement hors de son objet de classement.

b) Entités associées

Le type d'association CLASSEMENT associe de manière récursive le type d'entité INFORMATION.

Rôles : - "*est classé dans*" pour INFORMATION (1).
- "*dans lequel est classé*" pour INFORMATION (2).

Connectivités: - 0-1 pour INFORMATION (1).
- 0-N pour INFORMATION (2).

c) Attributs

Les attributs du type d'association CLASSEMENT sont les suivants:

- NUM_CLAS
  - cet attribut spécifie l'ordre de classement de l'objet informationnel qui participe à l'association par le rôle "*est classé dans*", à l'intérieur de l'objet informationnel structurant qui participe à l'association par le rôle "*dans lequel est classé*".
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est celui d'un nombre entier strictement positif.

d) Contraintes d'intégrité

Un message, un formulaire ou un document ne peut être classé que dans un dossier, un fichier ou une pile.

Une pile ne peut pas être classée dans un autre objet informationnel.

Un même objet informationnel ne peut participer que de manière exclusive aux types d'association CLASSEMENT, RANGEMENT et INTERFACE.

Un objet informationnel participe à au moins un des types d'association CLASSEMENT, RANGEMENT, TRAITEMENT et INTERFACE.

3) Le type d'association RANGEMENT

a) Description

Le type d'association RANGEMENT représente le rangement d'un objet informationnel élémentaire ou structurant dans un objet de rangement. Si un objet informationnel rangé participe à une association TRAITEMENT, alors il est physiquement hors de son objet de rangement.

b) Entités associées

Le type d'association RANGEMENT associe le type d'entité INFORMATION et le type d'entité OBJET_RANGEMENT.

Rôles : - "*est rangé dans*" pour INFORMATION.
- "*dans lequel est rangé*" pour OBJET_RANGEMENT.

Connectivités: - 0-1 pour INFORMATION.
- 0-N pour OBJET_RANGEMENT.

c) Contraintes d'intégrité

Aucun objet informationnel élémentaire ou structurant ne peut être rangé directement dans une armoire.

Un même objet informationnel ne peut participer que de manière exclusive aux types d'association RANGEMENT, CLASSEMENT et INTERFACE.

Un objet informationnel participe à au moins un des types d'association RANGEMENT, CLASSEMENT, TRAITEMENT et INTERFACE.

4) Le type d'association TRAITEMENT

a) Description

Le type d'association TRAITEMENT représente le traitement d'un O.I. sur un outil de travail. L'objet informationnel se trouve toujours logiquement dans son objet de rangement, dans son objet de classement ou dans son objet d'interface mais physiquement, il se trouve hors de celui-ci.

b) Entités associées

Le type d'association TRAITEMENT associe le type d'entité INFORMATION et le type d'entité OUTIL_DE_TRAVAIL.

Rôles : - "*est traité au moyen de*" pour INFORMATION.
- "*au moyen duquel est traité*" pour OUTIL_DE_TRAVAIL.

Connectivités: - 0-1 pour INFORMATION.
- 0-N pour OUTIL_DE_TRAVAIL.

c) Contraintes d'intégrité

Un objet informationnel participe à au moins un des types d'association TRAITEMENT, CLASSEMENT, RANGEMENT et INTERFACE.

5) Le type d'association INTERFACE

a) Description

Le type d'association INTERFACE représente la communication d'une information de ou vers le bureau. L'association indique alors qu'un objet informationnel se trouve dans un objet d'interface. Si un objet informationnel communiqué participe à une association TRAITEMENT, alors il est physiquement hors de son objet d'interface.

b) Entités associées

Le type d'association INTERFACE associe le type d'entité INFORMATION et le type d'entité OBJET_INTERFACE.

Rôles : - "*est transmis au moyen de*" pour INFORMATION.
- "*au moyen duquel est transmis*" pour OBJET_INTERFACE.

Connectivités: - 0-1 pour INFORMATION.
- 0-N pour OBJET_INTERFACE.

c) Contraintes d'intégrité

Seul une pile et une seule peut être associée à la boîte IN et à la boîte OUT. Cette pile leur est toujours associée, même lorsqu'elle est vide. Seul des messages peuvent être associés au téléphone.

Un même objet informationnel ne peut participer que de manière exclusive aux types d'association INTERFACE, CLASSEMENT et RANGEMENT.

Un objet informationnel participe à au moins un des types d'association INTERFACE, TRAITEMENT, CLASSEMENT, RANGEMENT.

6) Le type d'association COMPOSITION

a) Description

Le type d'association COMPOSITION représente la hiérarchisation des objets de rangement. Pour établir cette hiérarchie, les différentes contraintes exposées lors de la description de la classe objet de rangement, au point 3.4.3, sont utilisées. Le schéma de la figure 3.20 est donné uniquement pour modéliser cette hiérarchie; nous n'en donnerons donc pas la description.

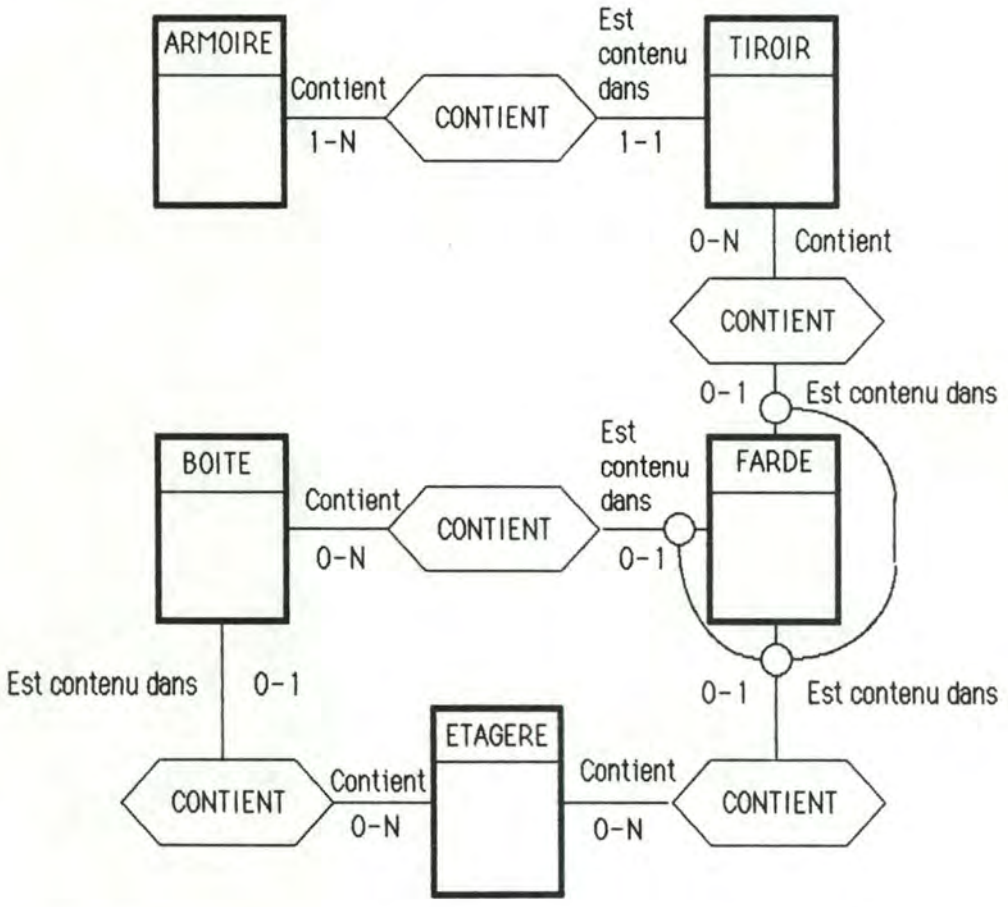


Fig 3.20 : Modélisation de la composition des objets de rangement

b) Entités associées

Le type d'association COMPOSITION associe de manière récursive le type d'entité OBJET_RANGEMENT.

Rôles : - "*est composant de*" pour OBJET_RANGEMENT (1).
- "*est composé de*" pour OBJET_RANGEMENT (2).

Connectivités: - 0-1 pour OBJET_RANGEMENT (1).
- 0-N pour OBJET_RANGEMENT (2).

c) Contraintes d'intégrité

Une armoire et une étagère ne peuvent être composant d'aucun autre objet de rangement.

Un tiroir ne peut exister sans être composant d'une armoire. Une farde ne peut exister sans être composant soit d'un tiroir, soit d'une boîte, soit d'une étagère.

Une armoire ne peut être composée que de tiroirs. Une étagère ne peut être composée que de boîtes ou de fardes. Un tiroir ou une boîte ne peut être composé que de fardes.

3.9 Les opérations primitives

Nous avons jusqu'à présent défini et analysé les objets qui se trouvent dans un bureau. Rappelons qu'un objet est constitué d'une information qui lui est propre et d'opérations primitives qu'on peut lui appliquer. Par la modélisation que nous venons d'exposer, on peut se rendre compte que la première moitié du travail a été réalisée.

L'objet de ce paragraphe va être de donner une description détaillée des opérations primitives que l'on peut exécuter sur les objets informationnels.

3.9.1 Introduction

Avant d'identifier les opérations primitives applicables aux objets informationnels relevés, il est nécessaire de connaître les différentes activités qui sont accomplies dans un bureau.

Une approche qui est fréquemment utilisée pour étudier ce qui se déroule dans un bureau, est le développement de taxonomies des tâches [HISA 84]. Une taxonomie des tâches est simplement une catégorisation des différentes activités qu'un travailleur peut réaliser.

La popularité d'une telle approche est due au fait qu'il est plus facile de déterminer l'impact d'une technologie quand le bureau est vu en termes d'activité plutôt qu'en termes de rôles ou de fonctions.

Pour établir de telles taxonomies, il est nécessaire de procéder à une observation du bureau afin de déterminer quelles sont les activités entreprises, le temps nécessaire à leur exécution et les procédures qui sont suivies pour les accomplir [HIRS 85]. Une comparaison entre de telles études est difficile, voire impossible, étant donné l'hétérogénéité de leurs hypothèses. Cependant, ces études sont faciles à mettre en oeuvre et de plus, lors de l'observation des activités de bureau, il y a peu de chances d'ambiguïté ou de mauvaise interprétation. Par exemple, si une personne parle au téléphone, ce fait est aisé à noter avec peu de chance qu'une autre personne le perçoive différemment.

3.9.2 Liste des opérations primitives

A partir d'une telle étude [HIRS 85], nous avons relevé en quelles opérations primitives sur les objets informationnels les activités observées se traduisaient.

Il est bon de rappeler qu'une telle étude n'est pas exhaustive dans son observation des activités de bureau et donc notre liste d'opérations primitives ne peut pas l'être non plus. Il pourra s'avérer utile de prévoir pour la suite du projet, un mécanisme d'ajout de nouvelles opérations primitives. Cependant, de par son aspect de prototype, nous n'envisagerons pas ce mécanisme dans notre travail.

Dans un premier temps, pour chaque objet informationnel, nous avons relevé les opérations primitives que l'on pouvait lui appliquer. Dans un deuxième temps, afin de pouvoir présenter des résultats plus synthétiques, nous avons regroupé par opération primitive, les objets informationnels auxquels elle pouvait être appliquée.

Le tableau 3.2 présente la liste des opérations primitives identifiées pouvant être appliquées aux objets informationnels élémentaires.

O.I.E / OPERATION	MESSAGE	FORMULAIRE	DOCUMENT
Création	X	X	X
Reproduction	X	X	X
Réception	X	X	X
Communication	X	X	X
Consultation	X	X	X
Modification			X
Complètement		X	
Vérification		X	
Destruction	X	X	X
Archivage		X	X
Classement	X	X	X
Rangement	X	X	X
Replacement	X	X	X
Décision		X	X

Tableau 3.2 : Liste des opérations primitives pour les O.I.E.

Le tableau 3.3 présente la liste des opérations primitives identifiées pouvant être appliquées aux objets informationnels structurants.

O.I.S / OPERATION	DOSSIER	FICHIER	PILE
Création	X	X	X
Réception	X	X	
Communication	X	X	
Feuilletage	X	X	X
Enlèvement			X
Vérification	X		
Destruction	X	X	X
Archivage	X	X	
Tri	X	X	X
Classement	X	X	
Rangement	X	X	X
Replacement	X	X	X
Décision	X		

Tableau 3.3 : Liste des opérations primitives pour les O.I.S.

3.10 Conclusion

Dans ce chapitre, nous avons exposé un premier modèle général des organisations que nous avons ensuite appliqué au bureau (point 3.1). Nous nous sommes limité dans ce point à ne prendre en considération que les tâches qui traitent de l'information.

Nous avons ensuite choisi de faire l'analyse du bureau selon l'approche des langages orientés objets (point 3.2) vu sa démarche très naturelle.

Avant de procéder à cette analyse, nous avons introduit les principaux concepts des langages orientés objets (point 3.3); analyse que nous avons détaillée ensuite (point 3.4). Nous avons relevé lors de celle-ci un certain nombre de concepts qui interviennent dans les tâches de bureau et qui correspondent à l'information et à la technologie.

Nous avons ensuite choisi de modéliser cette analyse grâce au modèle E/A (point 3.5) ce qui nous a amené à exprimer les concepts des langages orientés objets au moyen du modèle E/A (point 3.6).

Après avoir procédé à une abstraction sur notre modèle dû au fait que nous ne gérons pas de système d'information complet (point 3.7), nous avons exposé le schéma conceptuel du bureau (point 3.8).

Nous terminons ce chapitre par un relevé des opérations primitives possibles sur les différentes formes que prendra l'information dans un bureau (point 3.9). Il est notamment apparu qu'un mécanisme de création d'opérations primitives serait nécessaire pour la suite du projet.

Le chapitre suivant sera consacré à la description du langage que l'analyste pourra utiliser pour spécifier la tâche qu'il veut simuler.

Chapitre 4
Description du langage

CHAPITRE 4 : DESCRIPTION DU LANGAGE

4.1 Introduction

La description et la modélisation des objets de bureau étant données, nous définissons maintenant un outil qui va permettre à l'analyste de spécifier sa tâche.

Comme nous l'avons indiqué dans la description du diamant de Leavitt de la figure 3.2, une tâche est un enchaînement d'opérations primitives qui manipulent de l'information. L'outil de spécification que l'on va mettre au point doit donc permettre deux choses :

- premièrement, il doit donner à l'analyste la possibilité de spécifier chaque opération qui compose la tâche dans un formalisme dont la syntaxe est clairement définie. La spécification d'une opération consiste à donner, certaines informations qui vont la paramétriser.
- deuxièmement, pour respecter la définition de la tâche, il faut que cet outil offre à l'analyste des structures d'enchaînement qui lui permettront de mettre en relation les opérations qui forment la tâche. Les structures suivantes seront retenues : la séquence, la boucle et la condition.

Par rapport à ces deux exigences, un langage de spécification de haut niveau semble convenir.

4.1.1 Vue globale du langage

Notre langage est basé sur cinq types de composant :

- le type tâche,
- le type sous-schéma,
- le type condition,
- le type boucle,
- le type opération primitive.

La description d'une tâche se fera en décrivant un ensemble de composants appartenant à ces types; un composant étant décrit dans une phrase du langage. En outre, entre les composants, il y aura des relations de type déclenche qui permettront de créer des liens entre les composants. A l'exécution de la tâche, ces liens permettront de savoir dans quel ordre les composants seront exécutés.

L'exécution d'un composant signifie que le système va *retrouver* dans le système d'information, la phrase qui le décrit. Lorsqu'il l'aura trouvée, à partir des informations qui y sont mentionnées, il *réalisera* tout simplement ce que la phrase lui demande de faire.

Le schéma de la figure 4.1 permet d'avoir une première idée sur le fonctionnement du langage.

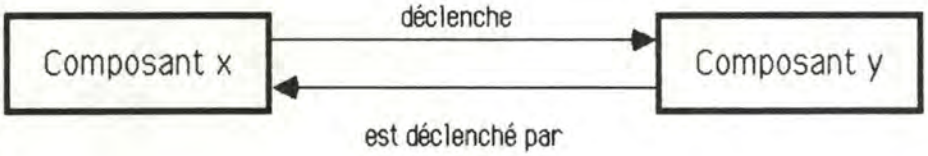


Fig. 4.1 : Les notions de composants et de déclenchement

Ceci ne constitue qu'une description globale du langage; la suite du chapitre est destiné à éclaircir chacunes de ces notions.

4.1.2 En quoi ce langage diffère-t-il des langages de programmation classiques ?

Dans un langage de programmation *classique* tel que COBOL, C, Pascal, ... une instruction est exécutée lorsque l'instruction qui la précède a terminé son exécution. Nous dirons que toute instruction est déclenchée implicitement par la fin de l'exécution de l'instruction précédente. Une telle philosophie exige de la part de l'utilisateur du langage, qu'il introduise les instructions dans l'ordre logique de son programme; c'est-à-dire l'ordre dans lequel elles seront exécutées. Ce type de langage est dit procédural.

Notre langage, quant à lui, se propose de rendre la notion de déclenchement explicite. En décrivant un composant dans une phrase, l'analyste devra spécifier quel autre composant du langage sera déclenché lorsque l'exécution du composant considéré sera terminée. Nous aurons donc un langage non procédural.

Cette philosophie a deux conséquences :

- tous les composants de la tâche doivent être décrits dans une phrase.
- tout composant décrit dans une tâche doit avoir un nom qui l'identifie parmi tous les composants de la tâche.

4.1.3 Quels sont les avantages d'un langage non procédural ?

Les notions de composant et de déclenchement permettent plusieurs choses :

- l'utilisateur n'est plus obligé d'introduire les composants qui décrivent sa tâche dans l'ordre où ils seront exécutés.
- puisque chaque composant est identifié, on peut envisager de mémoriser sa description dans une base de données. A partir de cette base de données, toute une série de rapports de documentation pourra être produite.
- à partir de la base de données, toute une série de tests de cohérence et

de complétudes pourra être réalisée.

Il nous reste donc :

- à présenter les conventions utilisées dans la description syntaxique des phrases du langages, (point 4.2)
- à présenter, pour toutes les opérations primitives, leur description et la syntaxe des phrases permettant de les décrire, (point 4.3)
- à présenter les différentes structures d'enchaînement des opérations primitives, (point 4.4)
- à présenter la construction d'une tâche à l'aide du concept de déclenchement, (point 4.5)
- à présenter le langage qui permet de définir l'environnement du bureau dans lequel la tâche commencera son exécution. (point 4.6)

4.2 Les conventions syntaxiques

Pour décrire la syntaxe des phrases de ce langage, nous utiliserons les conventions suivantes :

- les mots en majuscule sont des mots réservés du langage.
- les mots en minuscule sont des mots dont les valeurs sont choisies par l'utilisateur. Sauf remarque éventuelle, l'utilisateur aura la charge de faire commencer ces valeurs par une lettre.
- les mots entourés de crochets **[ ]** sont des clauses optionnelles à l'intérieur d'une construction.
- les mots entourés de parenthèses **( )** sont des clauses répétitives.
- les mots entourés d'accolades **{ }** et disposés en colonnes ne peuvent être utilisés simultanément mais au moins une des options doit être utilisée.

4.3 La description des opérations primitives

4.3.1 Introduction

Au paragraphe 3.9, nous avions fait le relevé des opérations primitives que l'on pouvait réaliser sur un objet informationnel en vue d'exécuter une tâche. Ces opérations primitives font parties du type de composant opération primitive.

Dans ce paragraphe, une descritpion détaillée de ces opérations est donnée. Dans cette description, on trouve la spécification et la syntaxe des phrases du langage qu'il faudra utiliser pour les décrire. Dans cette syntaxe, nous utiliserons les notions de variable et de paramètre; c'est pourquoi nous décrivons d'abord ces notions.

4.3.2 La notion de variable

La plupart des opérations primitives travaillent sur un ou plusieurs objets informationnels. Par exemple, détruire le message x. Dans ce cas, l'opération *détruire* travaille sur un objet dont l'identifiant est x. Cela exige que l'analyste, lorsqu'il décrit une opération de ce genre, précise l'identifiant de l'objet informationnel en question. Il pourra le faire de deux manières :

- soit respecter la syntaxe, décrite pour l'opération, pour introduire l'identifiant de l'objet. Prenons un exemple pour illustrer cette possibilité. Supposons que l'analyste veuille décrire une opération de suppression d'un formulaire de type *paie* et dont le nom est *dupont*. Il le fera en introduisant :

  DETRUIRE FORMULAIRE DE TYPE paie IDENTIFIE PAR dupont

- soit en utilisant une variable. Une variable porte un nom qui l'identifie dans l'ensemble des variables du système et référence l'identifiant d'un objet informationnel déterminé. Ainsi en reprenant l'exemple ci-dessus, on peut décider qu'il existe une variable Y qui référence l'identifiant du formulaire *dupont*. Dans ce cas, il suffira à l'analyste d'introduire :

  DETRUIRE VARIABLE y où y = "identifiant de l'objet informationnel à supprimer"

L'utilité de la variable est qu'elle va permettre à l'analyste de travailler sur des objets informationnels dont il ne connait pas l'identifiant. Nous verrons plus tard comment, il peut attribuer *un identifiant qu'il ne connait pas* à une variable (cfr. le type d'objet *boucle* au point 4.4.3 et l'opération primitive *enlever* au point 4.3.3 G).

4.3.3 La notion de paramètre

Un paramètre possède un nom qui l'identifie parmi tous les noms de paramètres qui existent dans le système. Il désigne un emplacement de la mémoire destiné à recevoir une réponse de l'utilisateur à l'écran. Un paramètre peut recevoir une valeur soit lors d'une opération de *vérification*, soit lors d'une opération de *décision*

4.3.4 La description des opérations primitives

Pour chaque opération primitive, nous décrirons :

- ses conditions d'applicabilité et ses implications sous forme de préconditions et de postconditions.
  La précondition est une condition qui doit expliciter les propriétés des arguments qui doivent être vérifiées avant toute exécution de l'opération primitive pour que celle-ci s'exécute correctement.

La postcondition est une condition qui doit expliciter les propriétés des résultats qui doivent être satisfaites à la fin de toute exécution de l'opération primitive si celle-ci s'est exécutée correctement.

- la syntaxe de la phrase qui permettra à l'analyste de la décrire. La phrase est l'ensemble des informations décrivant un composant de la tâche. Cette syntaxe permettra à l'analyste d'introduire les informations concernant les opérations primitives et devra être scrupuleusement respectée par celui-ci lorsqu'il introduira dans le système la spécification de sa tâche.

- les contraintes résultant des particularités du langage.

A) L'opération **créer**

- L'opération de création permet de créer un objet informationnel quelconque.

  Précondition
  - l'objet informationnel spécifié n'existe pas encore dans l'environnement de bureau.

  Postcondition
  - l'objet informationnel a été créé et se trouve associé à la table de travail.

- La phrase permettant de décrire une opération créer est la suivante :

```
CREER {MESSAGE nom_message}
      {FORMULAIRE DE TYPE type_formulaire IDENTIFIE PAR nom_formulaire}
      {DOCUMENT DE TYPE type_document IDENTIFIE PAR nom_document}
      {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
      {FICHIER nom_fichier}
      {PILE nom_pile}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[AVEC ORDRE DE CLASSEMENT = {ALPHABETIQUE}
                            {ALPHABETIQUE INVERSE}
                            {FIFO}
                            {LIFO}
                            {AUCUN}]
```

- La clause optionnelle spécifiant l'ordre de classement ne peut être utilisée que pour le dossier et le fichier. Pour la pile, l'ordre de classement pris en compte sera toujours le LIFO. La valeur par défaut pour cette clause est AUCUN.

B) L'opération **reproduire**

- L'opération de reproduction permet la reproduction en plusieurs exemplaires d'un objet informationnel élémentaire. Il est impossible de reproduire les objets informationnels structurants, à moins de reproduire chacun de leurs constituants.

  On peut faire des copies à partir d'une copie; dans ce cas les nouvelles copies seront considérées comme copie de l'original et non de la copie *copiée*.

  Préconditions
  - l'objet informationnel spécifié doit exister dans l'environnement de bureau et être un objet informationnel élémentaire.
  - il doit exister une photocopieuse dans l'environnement de bureau.
  - le numéro de départ doit être supérieur au numéro des copies déjà existantes pour l'original reproduit; même si on fait des copies d'une copie.

  Postconditions
  - les copies sont créées dans le système,
  - les copies sont associées à la photocopieuse,
  - si l'original existe alors les copies lui sont associées,
  - l'original copié ou la copie *copiée* est associée à la photocopieuse

- La phrase permettant de décrire une opération reproduire est la suivante :

```
REPRODUIRE [n FOIS] [A PARTIR DU NUMERO x]
      {[COPIE NUMERO n DE] MESSAGE nom_message}
      {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                           IDENTIFIE PAR nom_formulaire}
      {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                           IDENTIFIE PAR nom_document}
      {VARIABLE nom_variable}
```

- Si la clause optionnelle spécifiant le nombre de copies est omise, alors on reproduira à un seul exemplaire. Si la clause optionnelle spécifiant le numéro de départ pour les copies est omise, les copies seront numérotées à partir de un. Chaque copie aura pour ses attributs CLASSE, TYPE, NOM, les mêmes valeurs que celles de l'original. Si une variable est utilisée, elle ne peut référencer qu'un objet informationnel élémentaire.

C) L'opération **recevoir**

- L'opération de réception permet de recevoir un objet informationnel (à l'exception de la pile) par la boîte IN ou de recevoir un message par le téléphone. Par simplification, nous n'admettrons pas de recevoir un O.I.S. contenant d'autres O.I.S.

  Préconditions
  - lors de la réception par la boîte IN, l'objet informationnel spécifié ne doit pas déjà exister dans l'environnement de bureau. S'il s'agit d'un O.I.S., chacun de ses constituants ne doit pas déjà exister dans l'environnement de bureau.
  - lors de la réception par la boîte IN, il est nécessaire que celle-ci existe dans l'environnement de bureau et qu'une pile lui soit associée.
  - lors de la réception par le téléphone, il est nécessaire que celui-ci existe dans l'environnement de bureau. Si on désire garder une trace écrite de la communication téléphonique, alors le message sur lequel on garde la trace, ne doit pas déjà exister dans l'environnement de bureau.

  Postconditions
  - lors de la réception par la boîte IN d'un O.I.E., celui-ci a été créé et se trouve associé à la pile de la boîte IN.
  - lors de la réception par la boîte IN d'un O.I.S., celui-ci et chacun de ses constituants ont été créés et l'O.I.S. se trouve associé à la pile de la boîte IN. Chacun des constituants sont associés à leur O.I.S. par une association CLASSEMENT.
  - lors de la réception par le téléphone (avec trace écrite), le message a été créé et se trouve associé au téléphone.

- La phrase permettant de décrire une opération recevoir est la suivante :

  a) La réception d'un O.I.E. par le courrier intérieur ou extérieur

```
RECEVOIR [COPIE NUMERO n DE]
        {MESSAGE nom_message}
        {FORMULAIRE DE TYPE type_formulaire IDENTIFIE PAR nom_formulaire}
        {DOCUMENT DE TYPE type_document IDENTIFIE PAR nom_document}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
```

  b) La réception d'un dossier ou d'un fichier par le courrier intérieur ou extérieur

```
RECEVOIR {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
         {FICHIER nom_fichier}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[AVEC ORDRE DE CLASSEMENT = {ALPHABETIQUE}
```

{ALPHABETIQUE INVERSE}
{FIFO}
{LIFO}
{AUCUN}]
CONSTITUE DE [COPIE NUMERO n DE]
{MESSAGE nom_message}
{FORMULAIRE DE TYPE type_formulaire IDENTIFIE PAR nom_formulaire}
{DOCUMENT DE TYPE type_document IDENTIFIE PAR nom_document}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[(, DE [COPIE NUMERO n DE]
{MESSAGE nom_message}
{FORMULAIRE DE TYPE type_formulaire IDENTIFIE PAR nom_formulaire}
{DOCUMENT DE TYPE type_document IDENTIFIE PAR nom_document}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date])]

- Si l'ordre de classement spécifié pour l'objet informationnel structurant est ALPHABETIQUE ou ALPHABETIQUE INVERSE, on effectuera un tri des constituants. Si l'ordre de classement est LIFO ou FIFO, l'ordre spécifié dans l'énoncé est sensé représenter cet ordre.

c) La réception d'un message par le téléphone

RECEVOIR ORALEMENT MESSAGE
[AVEC TRACE ECRITE IDENTIFIEE PAR nom_message
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]]

D) L'opération **communiquer**

- L'opération de communication permet de communiquer un objet informationnel (à l'exception de la pile) par la boîte OUT ou de communiquer un message par le téléphone. Dans le cas de la communication téléphonique, il est possible de spécifier l'objet informationnel qui est à l'origine du message à communiquer.

Préconditions

- lors de la communication par la boîte OUT, l'objet informationnel doit exister dans l'environnement de bureau.
- lors de la communication par la boîte OUT, il est nécessaire que celle-ci existe dans l'environnement de bureau et qu'une pile lui soit associée.
- lors de la communication par le téléphone, il est nécessaire que celui-ci existe dans l'environnement de bureau. Si on a spécifié l'origine du message, celle-ci doit exister dans l'environnement de bureau.

Postconditions

- lors de la communication par la boîte OUT, l'objet informationnel est associé à la pile de la boîte OUT.
- lors de la communication téléphonique (avec source spécifiée), l'objet informationnel dont a été extrait le message retourne dans

son lieu de rangement ou de classement.
- si l'objet communiqué est un O.I.S., alors tous ses constituants sont aussi dans l'objet d'interface.

- La phrase permettant de décrire une opération communiquer est la suivante :

a) La communication par le courrier intérieur ou extérieur

```
COMMUNIQUER {[COPIE NUMERO n DE] MESSAGE nom_message}
            {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                                 IDENTIFIE PAR nom_formulaire}
            {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                                 IDENTIFIE PAR nom_document}
            {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
            {FICHIER nom_fichier}
            {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
```

- Si une variable est utilisée, elle ne peut pas référencer une pile.

b) La communication d'un message par le téléphone

```
COMMUNIQUER ORALEMENT MESSAGE
[A PARTIR DE {[COPIE NUMERO n DE] MESSAGE nom_message}
             {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                                  IDENTIFIE PAR nom_formulaire}
             {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                                  IDENTIFIE PAR nom_document}
             {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
             {FICHIER nom_fichier}
             {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date] ]
```

- Si une variable est utilisée, elle ne peut pas référencer une pile.

E) L'opération **consulter**

- L'opération de consultation permet de consulter le contenu informationnel d'un objet informationnel élémentaire.

Précondition
- l'objet informationnel spécifié doit exister dans l'environnement de bureau et être un objet informationnel élémentaire.

Postcondition
- l'objet informationnel a été consulté et se trouve associé à la table de travail.

- La phrase permettant de décrire une opération consulter est la suivante :

```
CONSULTER {[COPIE NUMERO n DE] MESSAGE nom_message}
          {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                               IDENTIFIE PAR nom_formulaire}
          {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                               IDENTIFIE PAR nom_document}
          {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
```

- Si une variable est utilisée, elle ne peut référencer qu'un objet informationnel élémentaire.

F) L'opération **feuilleter**

- L'opération de feuilletage permet de feuilleter un objet informationnel structurant et donc de connaître chacun des O.I.E. et O.I.S. qui en sont constituant. Tous les objets informationnels qui ne sont pas physiquement dans l'O.I.S. ne sont pas montrés lors de cette opération.

  Précondition
  - l'objet informationnel spécifié doit exister dans l'environnement de bureau et être un objet informationnel structurant.

  Postcondition
  - l'objet informationnel structurant a été feuilleté et se trouve associé à la table de travail. Des constituants de l'O.I.S. peuvent cependant se trouver sur un outil de travail.

- La phrase permettant de décrire une opération feuilleter est la suivante :

```
FEUILLETER {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
           {FICHIER nom_fichier}
           {PILE nom_pile}
           {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[AVEC ORDRE DE FEUILLETAGE = {ALPHABETIQUE}
                             {ALPHABETIQUE INVERSE}
                             {FIFO}
                             {LIFO}]
```

- Si une variable est utilisée, elle ne peut référencer qu'un objet informationnel structurant. L'analyste a la possibilité de spécifier un ordre de consultation. Cela n'influence en rien l'ordre de classement initial qui est attaché à l'objet informationnel structurant. Si aucun ordre de consultation n'est spécifié, alors l'ordre de classement de l'objet informationnel est utilisé.

G) L'opération **enlever**

- L'opération d'enlèvement permet d'enlever le premier élément d'une pile et de le placer sur le bureau.

  Précondition
  - la pile spécifiée doit exister dans l'environnement de bureau et ne pas être vide. Nous dirons qu'une pile est vide si physiquement, elle ne possède aucun constituant.

  Postcondition
  - le premier élément de la pile ne lui est plus associé et se trouve maintenant associé à la table de travail.

- La phrase permettant de décrire une opération enlever est la suivante :

  ENLEVER PREMIER CONSTITUANT nom_variable DE {PILE nom_pile}
  {VARIABLE nom_variable}

- Si une variable est utilisée, elle ne peut référencer qu'une pile. L'analyste aura à sa charge de vérifier que la pile n'est pas vide; cela est possible par l'intermédiaire de l'attribut ETAT_CONTENU.

  Avec cette opération, nous rencontrons la première possibilité d'affecter une valeur à une variable. En effet, cette opération a pour effet, non seulement, de placer le premier objet d'une pile en traitement sur la table de travail, mais aussi de placer son identifiant dans la variable dont le nom est spécifié.

H) L'opération **modifier**

- L'opération de modification permet de modifier un document et ce, éventuellement à partir d'un message brouillon.

  Préconditions
  - le document spécifié doit exister dans l'environnement de bureau.
  - si le message brouillon est spécifié, alors celui-ci doit exister dans l'environnement de bureau.

  Postconditions
  - le document a été modifié et se trouve associé à la table de travail.
  - le message brouillon éventuel est associé à la table de travail.

- La phrase permettant de décrire une opération modifier est la suivante :

```
MODIFIER   {[COPIE NUMERO DE] DOCUMENT DE TYPE type_document
                              IDENTIFIE PAR nom_document}
           {VARIABLE nom-variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[AU MOYEN DE [COPIE NUMERO n DE ] MESSAGE nom_message]
```

- Si une variable est utilisée, elle ne peut référencer qu'un document.

I) L'opération **compléter**

- L'opération de complètement permet de compléter un formulaire en remplissant les champs de celui-ci.

  Précondition
  - le formulaire spécifié doit exister dans l'environnement de bureau.

  Postcondition
  - le formulaire a été complété et se trouve associé à la table de travail.

- La phrase permettant de décrire une opération compléter est la suivante :

```
COMPLETER  {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                                IDENTIFIE PAR nom_formulaire}
           {VARIABLE nom_variable}
[AVEC ETAT = nom_état]  [AVEC ECHEANCE = date]
```

- Si une variable est utilisée, elle ne peut référencer qu'un formulaire.

J) L'opération **vérifier**

- L'opération de vérification permet de vérifier la complétude d'un dossier ou d'un formulaire. Un dossier est *complet* si toutes les pièces du dossier s'y trouvent. Un formulaire est *complet* si tous les champs sont remplis.

  Précondition
  - le dossier ou le formulaire spécifié doit exister dans l'environnement du bureau.

  Postconditions
  - l'objet informationnel se trouve associé à la table de travail.
  - la vérification a été faite et son résultat est connu du système.

- La phrase permettant de décrire une opération vérifier est la suivante :

```
VERIFIER {[COPIE NUMERO n DE ] FORMULAIRE DE TYPE type_formulaire
                                IDENTIFIE PAR nom_formulaire }
         {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
         {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[AVEC RESULTAT VERIFICATION nom_paramètre]
```

- Si une variable est utilisée, elle ne peut référencer qu'un formulaire ou un dossier. La réponse que l'utilisateur introduira affectera le paramètre dont le nom est spécifié. L'emploi de cette opération est la première des deux façons d'affecter une valeur à un paramètre. Plus tard, nous verrons que l'analyste pourra tester la valeur de ce paramètre.

K) L'opération **détruire**

- L'opération de destruction permet de détruire un objet informationnel quelconque en l'associant à la poubelle.

  Préconditions
  - l'objet informationnel spécifié doit exister dans l'environnement de bureau.
  - il doit exister une poubelle dans l'environnement de bureau.

  Postconditions
  - l'objet informationnel se trouve associé à la poubelle.
  - lors de la destruction d'un O.I.S., on peut considérer que tous ses constituants sont également détruits car ils lui restent associés.

- La phrase permettant de décrire une opération détruire est la suivante :

```
DETRUIRE {[COPIE NUMERO n DE] MESSAGE nom_message}
         {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                              IDENTIFIE PAR nom_formulaire}
         {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                              IDENTIFIE PAR nom_document}
         {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
         {FICHIER nom_fichier}
         {PILE nom_pile}
         {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
```

L) L'opération **archiver**

- L'opération d'archivage permet d'archiver un objet informationnel quelconque (à l'exception du message et de la pile) dans une boîte prévue à cet effet.

Préconditions

- l'objet informationnel spécifié doit exister dans l'environnement de bureau et ne peut être ni un message ni une pile.
- il doit exister une boîte à archives dans l'environnement de bureau.

Postconditions

- l'objet informationnel est associé à la boîte à archives.
- si l'objet archivé est un O.I.S. alors tous ses constituants se trouvent dans la boîte à archives.

- La phrase permettant de décrire une opération archiver est la suivante :

```
ARCHIVER {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                              IDENTIFIE PAR nom_formulaire}
         {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                              IDENTIFIE PAR nom_document}
         {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
         {FICHIER nom_fichier}
         {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
```

- Si une variable est utilisée, elle ne peut pas référencer un message ou une pile.

M) L'opération **trier**

- L'opération de tri permet de modifier l'ordre de classement d'un objet informationnel structurant. L'ordre de classement des constituants d'un O.I.S. est spécifié lors de la création de l'O.I.S. Nous prendrons en compte cinq ordres différents pour les O.I.S. : LIFO, FIFO, ALPHABETIQUE, ALPHABETIQUE INVERSE et AUCUN. Pour la pile, seul l'ordre de classement LIFO est possible.

Préconditions

- l'objet informationnel spécifié doit exister dans l'environnement de bureau et être un objet informationnel structurant.
- l'ordre de tri doit être un des quatre suivants : LIFO, FIFO, ALPHABETIQUE et ALPHABETIQUE INVERSE.

Postconditions

- les constituants de l'O.I.S. sont classés selon le nouvel ordre.
- dans le cas du dossier et du fichier, l'ordre de tri devient le nouvel ordre de classement. Dans le cas du tri d'une pile, les éléments de la pile sont triés selon l'ordre spécifié; mais par après, lors de l'insertion d'un élément dans la pile, celui-ci sera inséré selon l'ordre LIFO. Donc, on peut trier les éléments d'une pile mais l'ordre de classement reste LIFO.

- La phrase permettant de décrire une opération trier est la suivante :

```
TRIER {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
      {FICHIER nom_fichier}
      {PILE nom_pile}
      {VARIABLE nom_variable}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
SELON L'ORDRE {ALPHABETIQUE}
              {ALPHABETIQUE INVERSE}
              {FIFO}
              {LIFO}
```

- Si une variable est utilisée, elle ne peut référencer qu'un objet informationnel structurant.

N) L'opération **classer**

- L'opération de classement permet de classer un objet informationnel quelconque dans un objet informationnel structurant. La seule restriction étant qu'une pile ne peut être classée dans aucun O.I.S.

Préconditions
- l'objet informationnel spécifié comme étant à classer doit exister dans l'environnement de bureau de même que celui dans lequel on doit le classer.
- on ne peut pas classer une pile.

Postconditions
- l'objet informationnel à classer a été classé dans l'O.I.S. spécifié.
- si l'objet classé est un O.I.S. alors il se peut que certains de ses constituants soient en traitement sur un outil de travail.

- La phrase permettant de décrire une opération classer est la suivante :

a) Le classement d'un O.I.E. dans un O.I.S.

```
CLASSER {[COPIE NUMERO n DE] MESSAGE nom_message}
        {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                             IDENTIFIE PAR nom_formulaire}
        {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                             IDENTIFIE PAR nom_document}
        {VARIABLE nom_variable}
DANS {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
     {FICHIER nom_fichier}
     {PILE nom_pile}
```

- Si une variable est utilisée, elle ne peut référencer qu'un objet informationnel élémentaire.

b) Le classement d'un O.I.S. dans un autre O.I.S.

```
CLASSER {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
        {FICHIER nom_fichier}
        {VARIABLE nom_variable}
DANS {PILE nom_pile}
     {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
     {FICHIER nom_fichier}
```

- Si une variable est utilisée, elle ne peut référencer qu'un dossier ou un fichier.

O) L'opération **ranger**

- L'opération de rangement permet de ranger un objet informationnel quelconque dans un objet de rangement. La seule restriction est qu'une information ne peut être rangée directement dans une armoire mais uniquement par l'intermédiaire d'un de ses tiroirs.

  Préconditions
  - l'objet informationnel doit exister dans l'environnement de bureau de même que l'objet de rangement où il sera rangé.
  - on ne peut pas ranger un objet informationnel directement dans une armoire.

  Postcondition
  - l'objet informationnel à ranger a été rangé dans l'objet de rangement spécifié.

- La phrase permettant de décrire une opération ranger est la suivante :

```
RANGER {[COPIE NUMERO n DE] MESSAGE nom_message}
       {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                            IDENTIFIE PAR nom_formulaire}
       {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                            IDENTIFIE PAR nom_document}
       {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
       {FICHIER nom_fichier}
       {PILE nom_pile}
       {VARIABLE nom_variable}
DANS{ETAGERE nom_etagere}
    {BOITE nom_boite}
    {TIROIR nom_tiroir}
    {FARDE nom_farde}
```

P) L'opération **replacer**

- L'opération de replacement permet de remettre à leur place (soit lieu de classement, soit lieu de rangement) les objets informationnels qui se trouvent sur la table de travail. On pourra soit spécifier un objet

informationnel précis soit une classe d'objet informationnel soit tous les objets informationnels du bureau.

Précondition

- si un objet informationnel est spécifié comme étant à replacer, il doit exister dans l'environnement de bureau et se trouver sur la table de travail.

Postcondition

- les objets informationnels répondant au critère de sélection ont été replacés dans leur lieu de rangement, d'interface ou de classement.

- La phrase permettant de décrire une opération replacer est la suivante :

a) Le replacement d'un objet informationnel spécifié

```
REPLACER {[COPIE NUMERO n DE] MESSAGE nom_message}
         {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                              IDENTIFIE PAR nom_formulaire}
         {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                              IDENTIFIE PAR nom_document}
         {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
         {FICHIER nom_fichier}
         {PILE nom_pile}
         {VARIABLE nom_variable}
```

b) Le replacement de plusieurs objets informationnels selon un critère spécifié

```
REPLACER {TOUS LES OBJETS INFORMATIONNELS}
         {TOUS LES MESSAGES}
         {TOUS LES FORMULAIRES [DE TYPE type_formulaire]}
         {TOUS LES DOCUMENTS [DE TYPE type_document]}
         {TOUS LES DOSSIERS [DE TYPE type_dossier]}
         {TOUS LES FICHIERS}
         {TOUTES LES PILES}
```

Q) L'opération **décider**

- L'opération de décision permet de prendre une décision de routine sur base d'un formulaire, d'un document ou d'un dossier.

Précondition

- l'objet informationnel spécifié doit exister dans l'environnement de bureau et être un formulaire, un document ou un dossier.

Postconditions

- l'objet informationnel est associé à la table de travail.
- la décision a été prise et est connue du système.

- La phrase permettant de décrire une opération décider est la suivante :

```
DECIDER A PARTIR DE {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                                         IDENTIFIE PAR nom_formulaire}
                    {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                                         IDENTIFIE PAR nom_document}
                    {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
                    {VARIABLE nom_variable}
[AVEC RESULTAT DECISION nom_paramètre]
```

- Si une variable est utilisée, elle ne peut référencer qu'un formulaire, un document ou un dossier. La réponse que l'utilisateur introduira à l'écran affectera la valeur du paramètre spécifié. L'emploi de cette opération est la deuxième et dernière façon d'affecter une valeur à un paramètre. (la première façon était l'utilisation de l'opération vérifier).

4.4 Les structures d'enchaînement des opérations

Si nous nous en tenons à ce qui vient d'être décrit ci-dessus, on peut se rendre compte que nous sommes en présence de la description de plusieurs opérations qui n'ont aucuns liens entre elles. Graphiquement, cette situation pourrait se représenter de la façon suivante (cfr. figure 4.2).

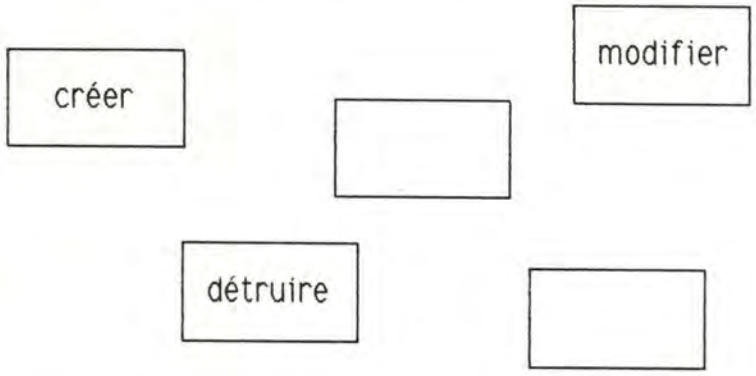


Fig. 4.2 : Des opérations primitives sans liens entre elles

L'objectif de l'introduction des structures est de créer un enchaînement entre ces opérations. Nous allons donc aborder les différentes structures que l'on peut trouver dans le langage. Pour chacune, nous la décrirons et nous en donnerons sa syntaxe.

4.4.1 La séquence

La séquence est certainement la structure la plus simple à décrire. On a une séquence à partir du moment où deux composants se suivent dans leur exécution. Il existe donc une relation de type déclenche entre ces composants.

4.4.2 La condition

La condition est un type de composant qui permet d'exprimer une alternative dans le déroulement de la tâche. Elle s'exprime généralement de la façon suivante : si une certaine clause est vérifiée, alors tel composant peut être exécuté, sinon tel autre composant est exécuté.

Une clause de condition exprime le type de condition que l'on veut utiliser. Par rapport à notre schéma de la figure 3.19, nous avons retenu trois clauses :

a) Test de la valeur d'un attribut

Le premier type de clause permet de tester la valeur d'un attribut d'un objet informationnel spécifié directement ou référencé par l'intermédiaire d'une variable.

Ce test se fera par rapport à une constante alphanumérique. Par simplification, nous n'avons pas admis la comparaison entre les attributs. Nous avons admis les six types habituels d'opérateurs de comparaison : [=, !=, <, >, <=, >=]

L'attribut ORDRE_CLAS spécifie l'ordre de classement des constituants d'un O.I.S. et l'attribut ETAT_CONTENU indique si un O.I.S. est vide ou non. De ce fait, on ne pourra tester la valeur d'un de ces attributs que dans le cas où l'objet spécifié est un O.I.S.

L'attribut NUM_COPIE spécifie le numéro de copie d'un O.I.E.; de ce fait, on ne pourra tester la valeur de cet attribut que si l'objet spécifié est un O.I.E.

```
SI {CLASSE}          DE {[COPIE NUMERO n DE] MESSAGE nom_message}
  {NOM}                 {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
  {TYPE}                                     IDENTIFIE PAR nom_formulaire}
  {ETAT}                {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
  {ECHEANCE}                                 IDENTIFIE PAR nom_document}
  {ORDRE_CLAS}       {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
  {ETAT_CONTENU}     {FICHIER nom_fichier}
  {NUM_COPIE}        {PILE nom_pile}
                     {VARIABLE nom_variable}

EST { = } "nom_constante_alpha-numérique"
    { != }
    { < }
    { > }
    { <= }
    { >= }
```

b) Test de l'appartenance d'un objet informationnel

Le deuxième type de clause permet de tester l'appartenance d'un objet informationnel, spécifié ou référencé par une variable, à un objet structurant donné qui peut lui même être référencé par une variable.

```
SI {[COPIE NUMERO n DE] MESSAGE nom_message}
  {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                       IDENTIFIE PAR nom_formulaire}
  {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                       IDENTIFIE PAR nom_document}
  {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
  {FICHIER nom_fichier}
  {VARIABLE nom_variable}

SE TROUVE DANS {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
               {FICHIER nom_fichier}
               {PILE nom_pile}
               {VARIABLE nom_variable}
```

c) Test de la valeur d'un paramètre

Le troisième type de clause permet de tester la valeur d'un paramètre.

```
SI PARAMETRE nom_paramètre { =  } "nom_constante_alpha-numérique"
                           { != }
                           { <  }
                           { >  }
                           { <= }
                           { >= }
```

Un paramètre étant destiné à recevoir une réponse de l'utilisateur lors d'une opération de vérification ou de décision, l'analyste grâce à cette clause va pouvoir en tester la valeur. Il pourra ainsi prévoir des traitements différents selon le résultat du test.

4.4.3 La boucle

La boucle est un type de composant qui va permettre de sélectionner un sous-ensemble d'objets informationnels.

Pour chaque boucle décrite dans une tâche, on va lui trouver associé :

- un critère,
- une variable,
- un corps.

Le critère d'une boucle exprime la condition qui va permettre de sélectionner des objets informationnels du système.

Lorsqu'un objet informationnel est sélectionné, son identifiant est affecté à la variable associée à la boucle. Ceci est le deuxième et dernier cas d'affectation d'une valeur à une variable. Pour rappel, le premier cas d'affectation était l'opération *enlever*.

Le corps d'une boucle est un ensemble de composants à exécuter pour chaque objet informationnel sélectionné. Si, dans ce corps, on trouve des opérations primitives, celles-ci pourront faire référence à la variable de la boucle. L'analyste peut ainsi manipuler des objets informationnels dont il ne connait pas l'identifiant. Par exemple, il pourrait demander tous les constituants d'un dossier. A partir de ces constituants et sans connaître leur identifiant, l'analyste pourra les manipuler et dire : tous ces constituants, dont je ne connait pas les identifiants, je veux les mettre sur la table de travail. Cette possibilité est la principale utilité des variables.

La philosophie de la boucle qui vient d'être exposée correspond tout à fait au concept de boucle développé dans le pseudo-langage ADL (Algorithm Description

Language) [HAIN 86]. Dans ce langage, une boucle va sélectionner dans une base de données, l'ensemble des *articles* qui satisfont une condition.

Par rapport à notre schéma de la figure 3.19, nous avons retenu trois critères de boucle possibles :

a) Sélection de tous les constituants d'un objet informationnel structurant

Ce critère permet de réaliser un traitement quelconque sur tous les objets faisant partie d'un O.I.S. L'analyste, à partir des constituants d'un O.I.S., a la possibilité de les choisir tous, de choisir uniquement ceux qui sont des O.I.E. ou encore de choisir uniquement ceux qui sont des O.I.S.

```
POUR CHAQUE CONSTITUANT {OIE}
                        {OIS}
                        {TOUS}
        DE  {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
            {FICHIER nom_fichier}
            {PILE nom_pile}
            {VARIABLE nom_variable}
```

b) Sélection de tous les objets informationnels d'une classe donnée et qui sont présents dans le système

Ce critère permet de sélectionner tous les objets informationnels d'une même classe.

```
POUR CHAQUE {MESSAGE}
            {FORMULAIRE}
            {DOCUMENT}
            {DOSSIER}
            {FICHIER}
            {PILE}
```

c) Sélection de tous les objets informationnels qui sont dans le système.

Ce critère permet de sélectionner tous les objets informationnels du système.

```
POUR CHAQUE OBJET INFORMATIONNEL
```

4.4.4 Boucle et condition imbriquées

Il est très intéressant de noter que si l'analyste décrit une boucle ou une condition dans sa tâche, celles-ci sont considérées comme étant des composants appartenant respectivement au type boucle et au type condition. Donc par définition du type boucle et du type condition, on peut dire que :

- une condition peut déclencher une autre condition,
- dans le corps d'une boucle, on peut trouver d'autres boucles.

4.5 La construction de la tâche

A ce stade de la description du langage, nous constatons que nous sommes en présence d'un certain nombre d'opérations primitives pouvant être reliées entre elles par des structures d'enchaînement. Cependant, nous ne savons toujours pas comment définir une tâche et quel sera l'ordre d'exécution des composants de la tâche.

Pour ce faire, nous allons introduire les types de composant tâche et sous-schéma et la notion de déclenchement.

Pour terminer, nous introduisons une représentation graphique pour chacun des types de composant vus ce qui donnera la possibilité à l'analyste de construire sa tâche sur papier avant de l'introduire dans le système à l'aide de notre langage.

4.5.1 La tâche

La notion de tâche n'est pas nouvelle et nous n'allons pas la redéfinir (cfr. le chapitre 3). Nous allons simplement dire en quoi l'introduction du type de composant tâche dans le langage est très intéressante.

Le type de composant tâche, dans le langage, a pour seule et unique fonction de rassembler l'ensemble des composants de la tâche sous un même nom : celui de la tâche. Cette possibilité de *regrouper* tous les composants de la tâche s'avèrera extrêmement utile lorsqu'il y aura plusieurs tâches décrites dans le système. Cette situation pourra se présenter lorsque la communication entre tâches sera gérée. Pour résoudre le problème de la communication, on peut très bien imaginer un langage, dont la philosophie serait très proche de notre langage et où on pourrait réaliser un enchaînement de tâches. Dès lors, on pourrait simuler une application de bureau qui se décomposerait en un enchaînement de tâches et chacune de ces tâches étant un enchaînement de composants.

4.5.2 Le sous-schéma

Le type de composant sous-schéma correspond à un enchaînement de composants qui ont un lien logique entre eux. Ainsi, les composants qui se trouvent dans le corps d'une boucle ont un lien logique entre eux : ils font partie du corps de la même boucle. En cela, nous pouvons dire que le corps d'une boucle constitue un sous-schéma. De la même manière, tous les composants d'une tâche ont un lien logique entre eux : ils font tous parties de la même tâche. En cela, nous pouvons dire que tous les composants d'une tâche, forment un sous-schéma.

La notion de sous-schéma correspond à la notion de bloc utilisée dans des langages tel que Pascal ou C. La seule différence réside dans leur utilisation. Dans le langage C, un bloc peut être associé à une condition, tandis que dans

notre langage cela n'est pas possible. Cela s'explique très facilement par le fait que dans un langage tel que C, les instructions ont un lien logique entre elles car elles font parties de la même condition. Mais aussi, et surtout, lorsque l'exécution du bloc est terminée, le programme reprend son cours normal. De cette manière, si le programme est bien construit, il n'y aura qu'un seul point de sortie au programme. Dans notre langage, cela est tout à fait différent car la condition n'est qu'un déroutement à l'intérieur de la tâche. Ainsi dans la structure de la tâche, il peut y avoir plusieurs points de sortie. Ces deux structures sont développées à la figure 4.3.

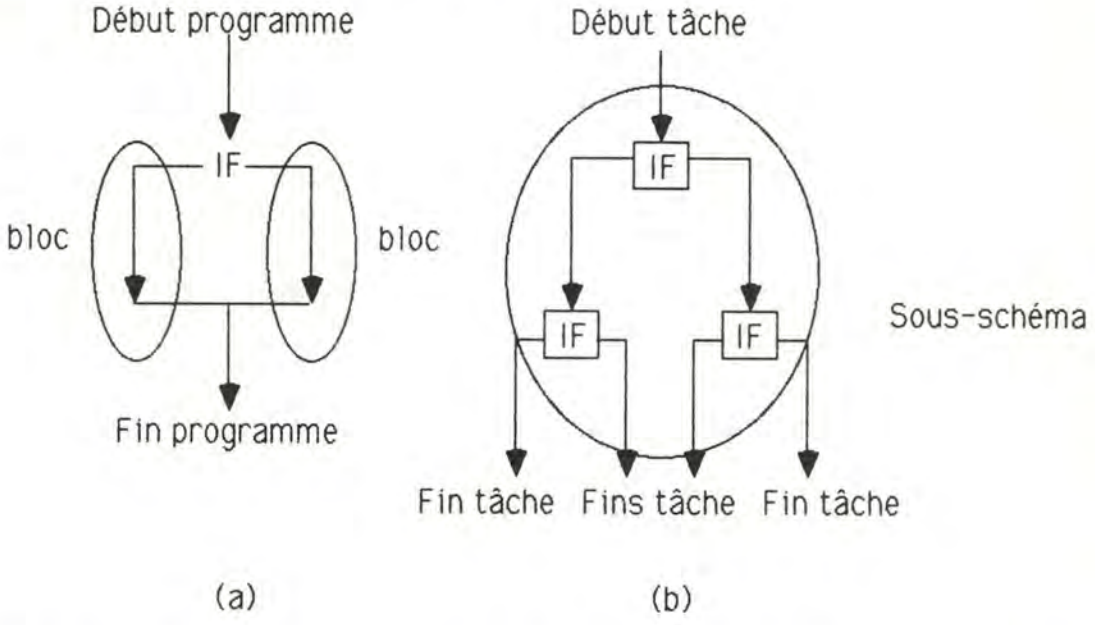


Fig. 4.3 : Différence d'utilisation entre a) la notion de bloc et b) la notion de sous-schéma

4.5.3 Le déclenchement

Le déclenchement d'un composant signifie que l'exécution du composant commence.

Dans la phrase décrivant un composant, il doit y avoir une clause qui spécifie quel composant sera déclenché lorsqu'il termine son exécution.

Lorsque l'analyste introduit une phrase décrivant un composant du langage, nous lui imposons de spécifié la forme passive de la clause déclenche , c'est-à-dire la clause *déclenché par*. L'utilisation obligatoire de cette clause permettra au système de détecter des fautes éventuellement commises par l'analyste. D'un autre côté, l'utilisation obligatoire de cette clause pourra également entraîner un surcroît d'erreurs ; des erreurs de frappe par exemple. Nous jugeons cependant préférable de détecter un maximum de fautes avant l'exécution de la simulation.

4.5.4 Construction de la tâche

Tous les types de composant du langage étant définis, nous allons décrire la syntaxe de la phrase qui permet de les spécifier. Pour toutes les phrase, la syntaxe est composée de quatre parties :

- l'identifiant du composant : il s'agit soit du nom de la tâche, soit du nom du sous-schéma, soit du nom de l'opération primitive, soit du nom de la condition ou soit du nom de la boucle. C'est grâce à cet identifiant que le système pourra *retrouver* dans le système d'information, la phrase qui décrit le composant.
- la description sous forme d'un texte du composant. (commentaire libre)
- l'identifiant du composant déclenché.
- l'identifiant du composant déclencheur.

Pour les boucles, les conditions et les opérations primitives, on trouvera en plus, ce qui a été décrit au point 4.3.3 pour les opérations primitives, au point 4.4.2 pour la condition et au point 4.4.3 pour la boucle.

Remarque

Précédement, nous avons dit que la condition créait à l'intérieur de la tâche un déroutement (cfr. figure 4.3. a). Cela implique qu'il peut y avoir plusieurs chemins à l'intérieur de la tâche et que, par conséquent, il est tout à fait possible que plusieurs chemins se rejoignent et déclenchent un même composant. Lors de la description de la tâche, il doit donc être possible de dire qu'un composant peut être déclenché par *plusieurs composants*. Il ne faut cependant pas perdre de vue que lors de l'exécution, la tâche emprunte un et un seul chemin; le composant ne pourra donc être effectivement déclenché que par un et un seul composant. De manière à rendre possible ce déclenchement multiple, la clause EST DECLENCHEE PAR est répetitive.

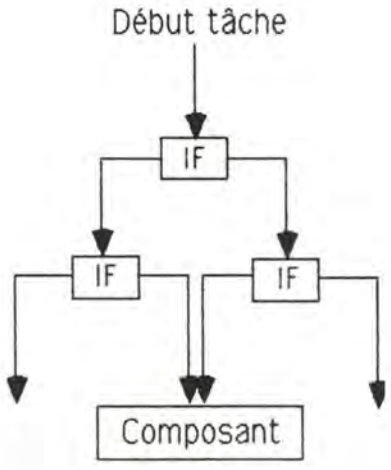


Fig. 4.4 : Au stade de la spécification, un composant peut être déclenché par plusieurs composants

A) La tâche

Les composants d'une tâche formant un sous-schéma, il est logique que la seule chose que puisse déclencher la tâche soit son sous-schéma. Le concept de tâche ne peut donc déclencher qu'un et un seul sous-schéma.

Le type de composant tâche étant le plus général, et ayant une fonction englobante, il ne pourra pas être déclenché par un autre type de composant. Dans la phrase décrivant une tâche, cela se traduit par l'absence de la clause DECLENCHE PAR.

TACHE nom_tache
[DESCRIPTION: texte]
DECLENCHE SOUS-SCHEMA nom_sous-schéma

B) Le sous-schéma

Un sous-schéma peut déclencher n'importe quel composant; à l'exception d'une tâche. Nous dirons que ce premier composant déclenché appartient au sous-schéma ainsi que le composant déclenché à la suite de ce premier composant et ainsi de suite. Le dernier composant appartenant au sous-schéma, pour en exprimer la fin, devra déclencher ce que l'on appelera la *fin-du-sous-schéma x*. Ce terme n'est en rien un nouveau type de composant mais tout simplement une notion qui remplit les fonctions suivantes :

- permettre au système de savoir qu'un sous-schéma est terminé.
- permettre à l'analyste de mieux se rendre compte de la fin d'un sous-schéma.

En tenant compte de ce qui vient d'être dit, nous dirons qu'un sous-schéma peut en contenir un autre. Cette situation se produira lorsqu'il y aura une boucle dans la tâche. Si cette imbrication est faisable, elle doit cependant rester logique; un composant appartenant à un et un seul sous-schéma, celui-ci ne peut déclenché la fin que de son sous-schéma.

Puisque le corps d'une boucle est un sous-schéma et qu'une tâche est un sous-schéma, tout sous-schéma ne peut être déclenché que par une tâche ou par une boucle.

SOUS-SCHEMA nom_sous-schéma
[DESCRIPTION: texte]
DECLENCHE PAR {TACHE nom_tache}
{BOUCLE nom_boucle}
DECLENCHE {OPERATION PRIMITIVE nom_opération_primitive}
{CONDITION nom_condition}
{BOUCLE nom_boucle}

C) La boucle

Tant que tous les objets informationnels sélectionnés par la boucle n'ont pas été traités, la boucle va déclencher son sous-schéma. Lorsqu'ils seront tous traités, la boucle pourra déclencher soit :

- un autre composant à l'exception d'une tâche,
- la fin du sous-schéma initial si la boucle est le dernier composant de la tâche exécuter.

Une boucle peut être déclenchée par un et un seul sous-schéma et/ou par un ou plusieurs composants.

```
BOUCLE nom_boucle
[DESCRIPTION: texte]
ENONCE: énoncé_boucle
DECLENCHEE PAR {SOUS-SCHEMA nom_sous-schéma}
               {OPERATION PRIMITIVE nom_opération_primitive}
               {CONDITION nom_condition}
               {BOUCLE nom_boucle}
             [(,{OPERATION PRIMITIVE nom_opération_primitive}
               {CONDITION nom_condition}
               {BOUCLE nom_boucle} )]
POUR CHAQUE occurrence nom_variable
DECLENCHE SOUS-SCHEMA nom_sous-schéma
ENSUITE DECLENCHE {OPERATION PRIMITIVE nom_opération_primitive}
                  {CONDITION nom_condition}
                  {BOUCLE nom_boucle}
                  {FIN-SOUS-SCHEMA nom_sous-schéma}
```

D) La condition

D'après la valeur booléenne de la condition, un composant est déclenché si la condition est vraie ou un autre composant est déclenché si la condition est fausse. Si la condition est le dernier composant exécuté de la tâche, alors elle peut déclencher la fin du sous-schéma initial.

La condition peut être déclenchée par un et un seul sous-schéma et/ou par un ou plusieurs composants.

```
CONDITION nom_condition
[DESCRIPTION: texte]
ENONCE: énoncé_condition
DECLENCHEE PAR {SOUS-SCHEMA nom_sous-schéma}
               {OPERATION PRIMITIVE nom_opération_primitive}
               {CONDITION nom_condition}
               {BOUCLE nom_boucle}
             [(,{OPERATION PRIMITIVE nom_opération_primitive}
               {CONDITION nom_condition}
               {BOUCLE nom_boucle} )]
SI VRAI DECLENCHE {OPERATION PRIMITIVE nom_opération_primitive}
                  {CONDITION nom_condition}
```

{BOUCLE nom_boucle}
{FIN-SOUS-SCHEMA nom_sous-schéma}
SI FAUX DECLENCHE {OPERATION PRIMITIVE nom_opération_primitive}
{CONDITION nom_condition}
{BOUCLE nom_boucle}
{FIN-SOUS-SCHEMA nom_sous-schéma}

E) L'opération primitive

Une opération primitive peut déclencher n'importe quel composant; à l'exception d'une tâche et d'un sou-schéma. Elle peut aussi déclencher la fin du sous-schéma initial si elle constitue le dernier composant exécuté de la tâche.

Une opération primitive peut être déclenchée par un et un seul sous-schéma ou par un ou plusieurs composants.

OPERATION PRIMITIVE nom_opération_primitive
[DESCRIPTION : texte]
ENONCE: énoncé_opération_primitive
DECLENCHEE PAR {SOUS-SCHEMA nom_sous-schéma}
{OPERATION PRIMITIVE nom_opération_primitive}
{CONDITION nom_condition}
{BOUCLE nom_boucle}
[(,{OPERATION PRIMITIVE nom_opération_primitive}
{CONDITION nom_condition}
{BOUCLE nom_boucle})]
DECLENCHE {OPERATION PRIMITIVE nom_opération_primitive}
{CONDITION nom_condition}
{BOUCLE nom_boucle}
{FIN-SOUS-SCHEMA nom_sous-schéma}

4.5.5 Représentation graphique

Ce que nous allons présenter maintenant ne doit surtout pas être vu comme un moyen d'éviter les erreurs dans la description d'une tâche. En effet, il a été montré que l'utilisation de diagrammes dans la résolution d'un problème ne conduit pas nécessairement à une solution correcte [SHNE 77].

Cependant, certaines personnes sont convaincues au contraire qu'une représentation schématisée peut être très utile. C'est pourquoi, nous proposons ci-dessous, une schématisation des types de composant pour pouvoir structurer la tâche à simuler sur papier avant de l'introduire dans le système via notre langage.

- Tâche → Nom tâche

- Sous-schéma → Nom sous-schéma

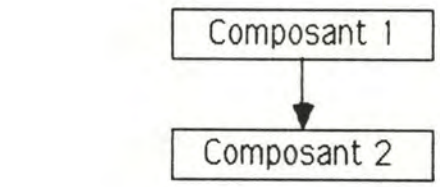
- Condition, boucle, opération primitive → Nom composant

La notion de déclenchement sera représentée par :

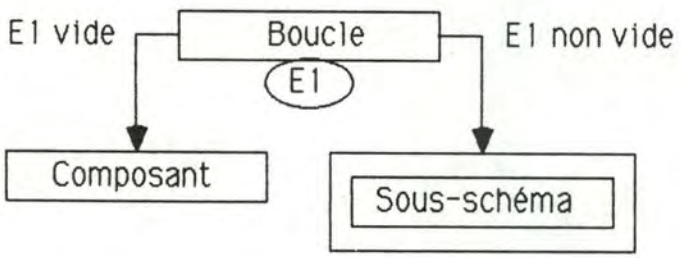
→

Les structures de la dynamique auront la représentation suivante :

- Séquentiel =

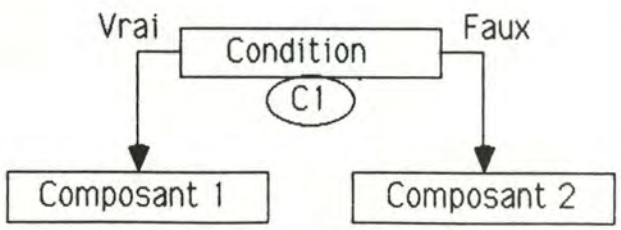


- Boucle =



E1 représente le sous-ensemble des objets informationnels sélectionné. Sous le schéma de la tâche, il faudra indiquer le critère de sélection correspondant à *E1*.

- Condition =



C1 représente la condition. Sous le schéma de la tâche, il faudra indiquer la condition correspondant à *C1*.

A l'aide de cette représentation, la structure d'une tâche peut se résumer par le schéma de la figure 4.5.

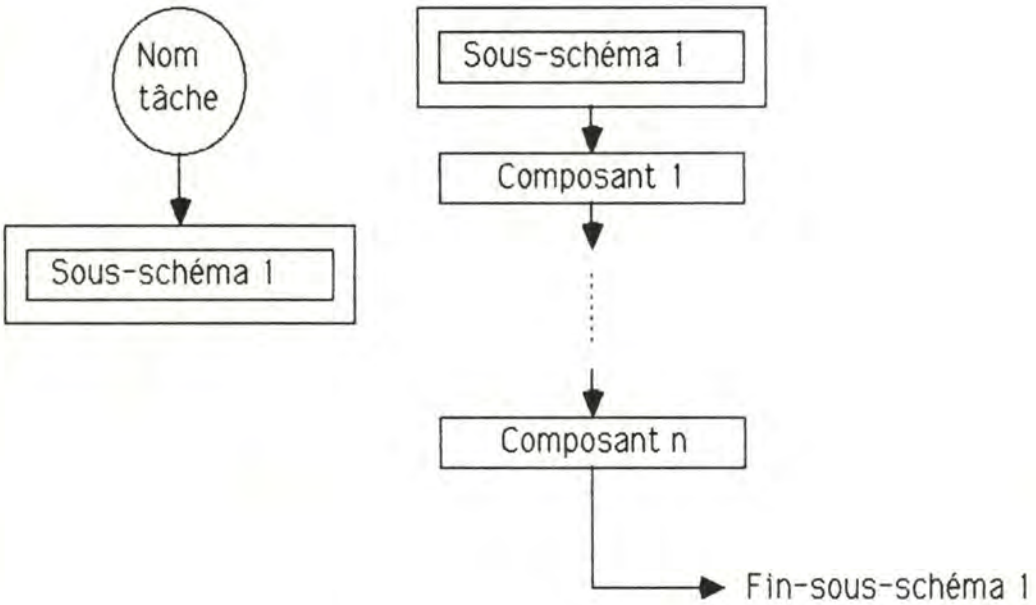


Fig. 4.5 : Schéma de la tâche

4.6 La déclaration de l'environnement

Avant de pouvoir exécuter une tâche, il est nécessaire d'avoir défini l'environnement de départ. Cette définition consiste d'une part à identifier les différents objets qui existent dans le bureau et d'autre part à spécifier quelles sont les associations qui peuvent exister entre ces objets. Par exemple, on doit pouvoir créer un dossier que l'on range dans un tiroir.

Remarques

- Précédement, nous avons signalé que l'existence d'une et un seule table de travail était obligatoire et que dans les boîtes IN et OUT, il y avait toujours une pile. De façon à ce que l'analyste se rende compte de leur existence, nous lui demanderons de créer une table de travail et deux objets piles associés chacun respectivement aux boîtes IN et OUT.
- En général pour tous les objets, l'analyste a le choix du nom qu'il désire leurs donner; excepté cependant pour certains dont le nom est donné par le système :

  - Téléphone -> le nom = téléphone
  - Poubelle -> le nom = poubelle
  - Boîte (IN) -> le nom = in

- Boîte (OUT) -> le nom = out
- Boîte (archive) -> le nom = archive

4.6.1 La déclaration des objets

Pour rappel, le modèle est basé sur quatre super-classes d'objet :

- les objets informationnels,
- les objets de rangement,
- les outils de travail,
- les objets d'interface.

A) La déclaration d'un objet informationnel

```
CREER OBJET INFORMATIONNEL
      {[COPIE NUMERO n DE] MESSAGE nom_message}
      {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                           IDENTIFIE PAR nom_formulaire}
      {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                           IDENTIFIE PAR nom_document}
      {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
      {FICHIER nom_fichier}
      {PILE nom_pile}
[AVEC ETAT = nom_état] [AVEC ECHEANCE = date]
[AVEC ORDRE DE CLASSEMENT = {ALPHABETIQUE}
                            {ALPHABETIQUE INVERSE}
                            {FIFO}
                            {LIFO}
                            {AUCUN}]
```

B) La déclaration d'un objet de rangement

```
CREER OBJET RANGEMENT
      {ARMOIRE nom_armoire}
      {ETAGERE nom_étagère}
      {FARDE nom_farde}
      {TIROIR nom_tiroir}
      {BOITE nom_boite}
      {BOITE ARCHIVE}
```

C) La déclaration d'un outil de travail

```
CREER OBJET TRAITEMENT
      {TABLE DE TRAVAIL}
      {PHOTOCOPIEUSE}
```

D) La déclaration d'un objet d'interface

```
CREER OBJET INTERFACE
      {TELEPHONE}
      {POUBELLE}
      {BOITE IN}
```

{BOITE OUT}

4.6.2 La déclaration des associations entre les objets

Pour rappel, il y a quatre types d'association entre les objets pouvant exister avant le début de la tâche :

- un objet informationnel *est rangé dans* un objet de rangement.
- un objet informationnel *est classé dans* un objet informationnel structurant.
- un objet informationnel *est transmis au moyen* d'un objet d'interface.
- un objet de rangement *est composé de* un ou plusieurs objets de rangement.

L'association *copie* sera créée directement par le système entre les O.I.E. qui sont d'une même classe, d'un même type et qui un nom identique.

A) Un objet informationnel est rangé dans un objet de rangement

```
CREER ASSOCIATION RANGEMENT ENTRE
      {[COPIE NUMERO n DE] MESSAGE nom_message}
      {[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
                           IDENTIFIE PAR nom_formulaire}
      {[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
                           IDENTIFIE PAR nom_document}
      {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
      {FICHIER nom_fichier}
      {PILE nom_pile}
ET {ETAGERE nom_étagère}
   {BOITE nom_boîte}
   {TIROIR nom_tiroir}
   {FARDE nom_farde}
```

B) Un objet informationnel est classé dans un O.I.S.

1. Le classement d'un O.I.E. dans un O.I.S.

```
CREER ASSOCIATION CLASSEMENT ENTRE [COPIE NUMERO n DE]
      {MESSAGE nom_message}
      {FORMULAIRE DE TYPE type_formulaire IDENTIFIE PAR nom_formulaire}
      {DOCUMENT DE TYPE type_document IDENTIFIE PAR nom_document}
ET {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
   {FICHIER nom_fichier}
   {PILE nom_pile}
```

2. Le classement d'un O.I.S. dans un autre O.I.S.

```
CREER ASSOCIATION CLASSEMENT ENTRE
      {DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
      {FICHIER nom_fichier}
ET {PILE nom_pile}
```

{DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
{FICHIER nom_fichier}

C) Un objet informationnel est transmis au moyen d'un objet d'interface

CREER ASSOCIATION INTERFACE ENTRE
{[COPIE NUMERO n DE] MESSAGE nom_message}
{[COPIE NUMERO n DE] FORMULAIRE DE TYPE type_formulaire
IDENTIFIE PAR nom_formulaire}
{[COPIE NUMERO n DE] DOCUMENT DE TYPE type_document
IDENTIFIE PAR nom_document}
{DOSSIER DE TYPE type_dossier IDENTIFIE PAR nom_dossier}
{FICHIER nom_fichier}
{PILE nom_pile}
ET {BOITE IN}
{BOITE OUT}
{TELEPHONE}
{POUBELLE}

D) Un objet de rangement est composé de un ou plusieurs autres objets de rangement

CREER ASSOCIATION COMPOSITION ENTRE
{TIROIR nom_tiroir}
{FARDE nom_farde}
{BOITE nom_boite}
ET {ARMOIRE nom_armoire}
{TIROIR nom_tiroir}
{ETAGERE nom_étagère}
{BOITE nom_boîte}

4.7 Conclusion

Ce chapitre avait pour objet la présentation du langage que l'analyste utilisera pour décrire la tâche qu'il veut simuler.

Ce langage est un langage non procédural basé sur cinq types de composant pouvant être reliés par des relations de déclenchement. La tâche est décrite comme un ensemble de composants reliés entre eux par de telles relations.

La relation de déclenchement entre les composants exprime la notion de séquence d'exécution : l'exécution d'un composant est déclenchée par la fin de l'exécution du composant qui le déclenche.

On trouvera à l'annexe 1, un exemple d'application du langage à un cas réel.

Le chapitre suivant sera consacré à la modélisation de ce langage; nous y verrons comment on peut représenter toutes les notions présentées dans le schéma Entité-Association.

Chapitre 5
Modélisation du langage

CHAPITRE 5 : MODELISATION DU LANGAGE

5.1 Introduction

L'information introduite par l'analyste au moyen du langage que nous venons de décrire, devra être exploitée par le système de manière à pouvoir simuler graphiquement la tâche décrite. Cette information se trouve dans les différentes phrases que l'analyste a déclarées et qui se trouvent *pêle-mêle* mémorisées sur un support quelconque. (pour rappel, l'analyste peut spécifier les composants de la tâche dans n'importe quel ordre).

A partir de cet ensemble de phrases, nous allons en extraire les informations utiles à la simulation et nous les mémoriserons dans une base de données : la base de données du langage. Dans celle-ci, nous trouverons :

- toutes les relations de déclenchement qui peuvent exister entre ces différents composants et qui ont été déclarées par l'analyste (point 5.2),
- l'information décrite dans les phrases (point 5.3).

5.2 Modélisation de la notion de déclenchement et des types de composant

5.2.1 Schéma conceptuel de la base de données du langage

Dans le schéma conceptuel (cfr. fig. 5.1) qui modélise la base de données du langage, les cinq composants de base du langage (tâche, sous-schéma, boucle, condition, opération primitive) sont représentés sous forme de types d'entité.

Toutes les clauses du langage du type DECLENCHE et du type DECLENCHEE PAR sont modélisées par un type d'association entre le type d'entité déclencheur et le type d'entité déclenché.

Sur ce schéma, on peut voir aisément les restrictions qu'il y a au niveau des composants qui peuvent en déclencher d'autres. Par exemple, une tâche ne peut pas déclencher de boucle, ou encore une condition ne peut pas déclencher un sous-schéma.

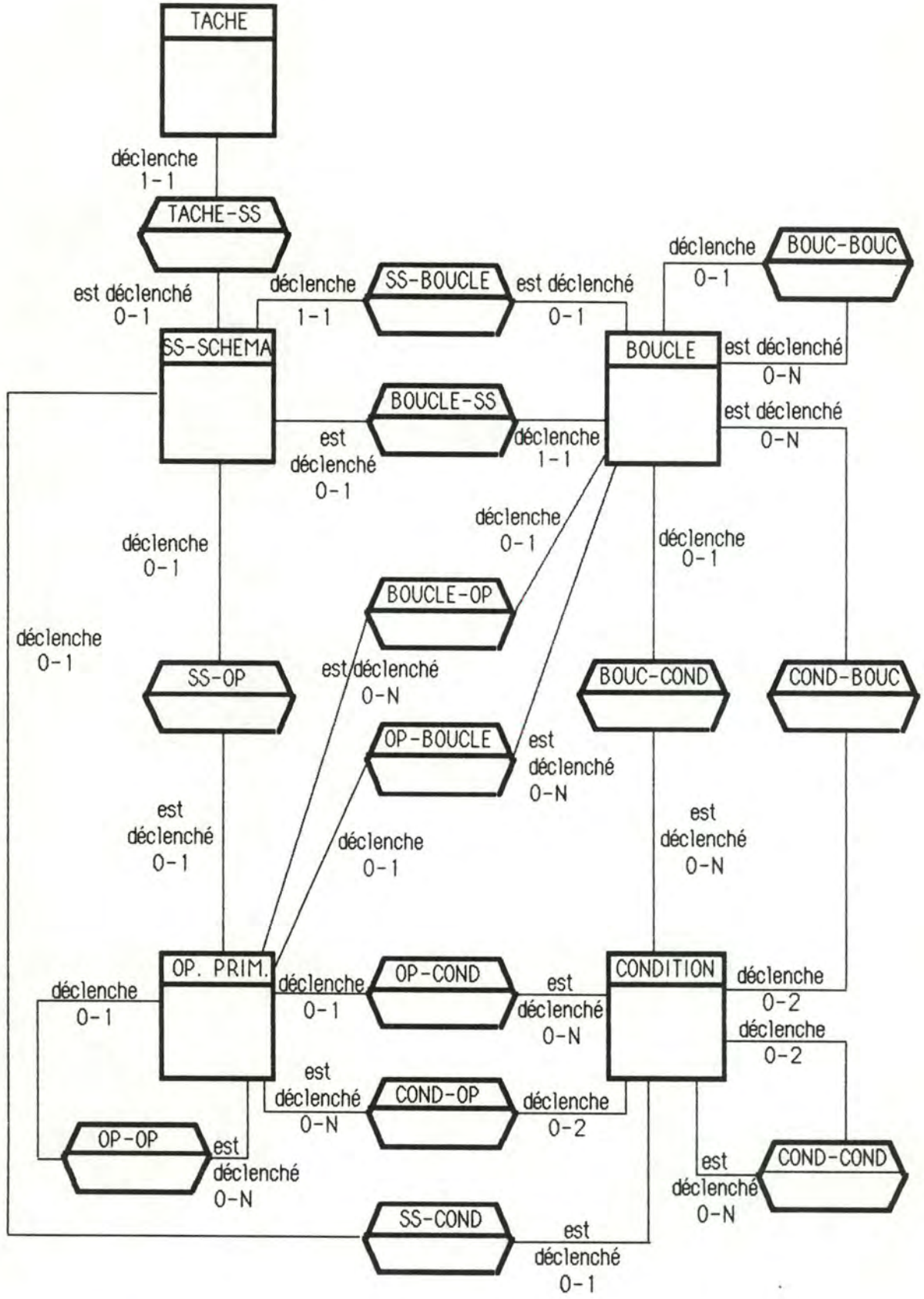


Fig. 5.1 : Schéma conceptuel modélisant la dynamique du langage

Remarque

Une condition peut être évaluée à la valeur VRAI ou à la valeur FAUX. Ainsi, dans chacun des cas, un composant différent peut être déclenché.

Cela explique la connectivité 0-2 pour toutes les associations où la condition participe par le rôle *déclenche*.

Il reste cependant une notion du langage à représenter; il s'agit du déclenchement de la fin d'un sous-schéma.

Exemple : OPERATION PRIMITIVE ...
DECLENCHE FIN_SOUS_SCHEMA nom_sous_schéma

Il existe donc une relation particulière entre le type d'entité OPERATION_PRIMITIVE et le type d'entité SOUS_SCHEMA. De façon générale, cette relation existe aussi entre le type d'entité BOUCLE et le type d'entité SOUS_SCHEMA; elle existe aussi entre le type d'entité CONDITION et le type d'entité SOUS_SCHEMA. En effet, nous pouvons avoir :

BOUCLE ...
DECLENCHE FIN_SOUS_SCHEMA nom_sous_schéma

CONDITION ...
DECLENCHE FIN_SOUS_SCHEMA nom_sous_schéma

Cette relation sera modélisée sous la forme de type d'association entre les types d'entités concernés. De manière à ne pas surcharger le schéma conceptuel de la figure 5.1, nous en avons repris les éléments intervenants dans ces associations dans la figure 5.2 :

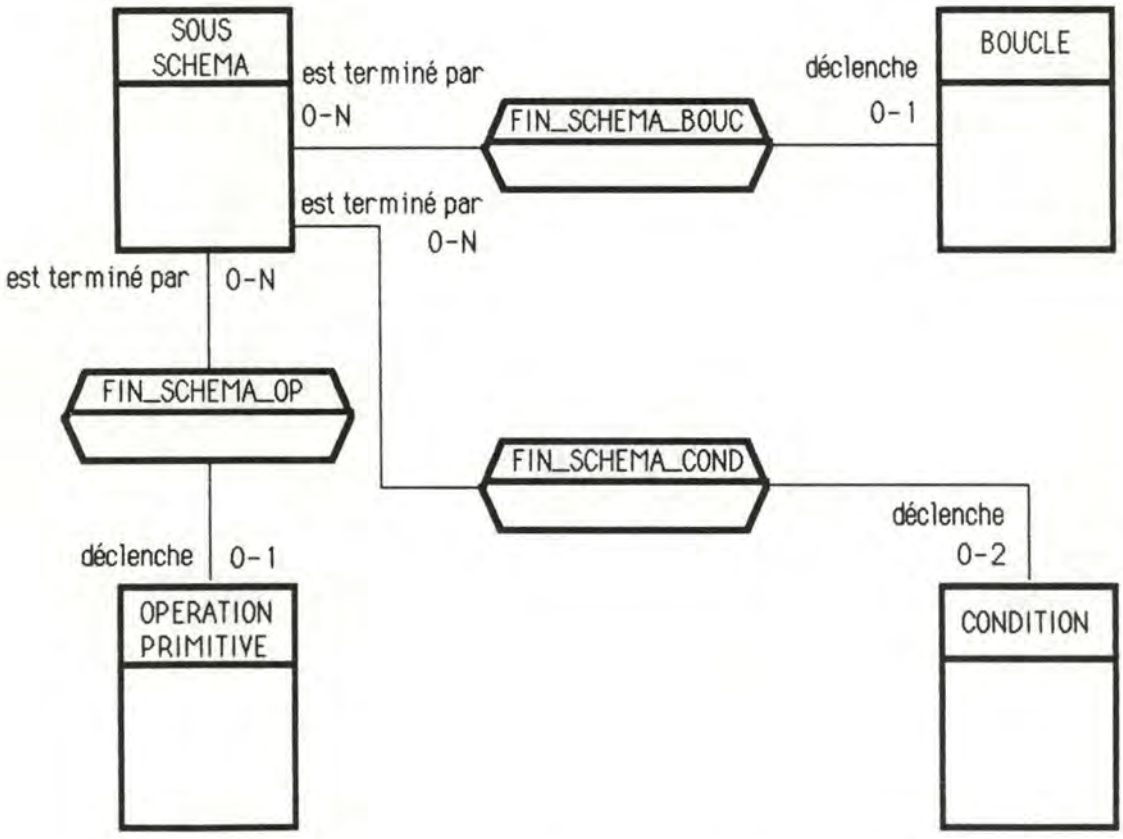


Fig. 5.2 : Modélisation du déclenchement de la fin d'un sous-schéma

Remarque

La fin d'un sous-schéma peut être déclenchée par plusieurs composants. Cela résulte d'une remarque faite précédement : dans l'enchaînement de la tâche, il peut y avoir plusieurs chemins qui se rejoignent lors du déclenchement d'un composant ou de la fin d'un sous-schéma.

Dans le cadre de ce mémoire, toute tâche n'est déclenchée par aucun autre composant. Nous pouvons le constater par l'absence de clause DECLENCHEE PAR dans la syntaxe de la phrase qui décrit une tâche au point 4.5.4 A.

Cela se traduit sur le schéma par le fait que l'entité TACHE ne participe à aucune association par le rôle *est déclenché*. En tenant compte de cette remarque, nous pouvons exprimer la règle générale suivante :

> Toutes occurrences des types d'entité TACHE, SOUS-SCHEMA, CONDITION, BOUCLE et OPERATION_PRIMITIVE participent toujours à au moins une occurrence d'association par le rôle *déclenche*.
> Toutes occurrences des types d'entité SOUS-SCHEMA, CONDITION, BOUCLE et OPERATION_PRIMITIVE participent toujours à au moins une occurrence d'association par le rôle *est déclenché*.

Cela signifie que tous les composants décrits dans une tâche d'une part sont déclenchés par au moins un composant et d'autre part déclenchent au moins un composant.

5.2.2 Description des types d'association

5.2.2.1 Description des types d'association de la figure 5.1

a) Description

Les types d'association de la figure 5.1 modélisent les clauses de déclenchement intervenant entre les différents types de composant de notre langage, à l'exception des relations de déclenchement de fin de sous-schéma.

b) Entités associées

types d'entité / types d'association	TACHE	SOUS SCHEMA	BOUCLE	CONDITION	OPERATION PRIMITIVE
TACHE-SS	déclenche 1-1	est déclenché 0-1			
SS-BOUCLE		déclenche 0-1	est déclenché 0-1		
SS-COND		déclenche 0-1		est déclenché 0-1	
SS-OP		déclenche 0-1			est déclenché 0-1
BOUCLE-SS		est déclenché 0-1	déclenche 1-1		
BOUC-BOUC			déclenche 0-1 est déclenché 0-N		
BOUC-COND			déclenche 0-1	est déclenché 0-N	
BOUCLE-OP			declenche 0-1		est déclenché 0-N
COND-BOUC			est déclenché 0-N	déclenche 0-2	
COND-COND				déclenche 0-2 est déclenché 0-N	
COND-OP				déclenche 0-2	est déclenché 0-N
OP-BOUCLE			est déclenché 0-N		déclenche 0-1
OP-COND				est déclenché 0-N	déclenche 0-1
OP-OP					déclenche 0-1 est déclenché 0-N

Tableau 5.1 : Rôles et connectivités des types d'entité associés

Le tableau 5.1 indique pour chaque type d'association, quels types d'entité il associe. Pour chaque type d'entité associé, on indique son nom de rôle et sa connectivité.

c) Attributs

Seul les associations auxquelles la condition participe par le rôle *déclenche* ont un attribut. En voici, la description :

- EVALUATION
  - l'évaluation d'une condition étant un résultat booléen, cet

attribut indique quel composant sera déclenché si le résultat est vrai et quel composant sera déclenché si le résultat est faux.
- attribut de type élémentaire, obligatoire et simple.
- le domaine de valeurs est l'ensemble suivant : {VRAI, FAUX}.

d) Contraintes d'intégrité

Un sous-schéma doit participer à une et une seule association par le rôle *est déclenché* et à une et une seule association par le rôle *déclenche*.

Une boucle doit participer à au moins une association par le rôle *est déclenché* et à au plus deux associations par le rôle *déclenche*.

Une condition doit participer à au moins une association par le rôle *est déclenché* et à deux associations par le rôle *déclenche*. Ces deux associations doivent avoir des valeurs différentes pour leur attribut EVALUATION : ces valeurs sont soit VRAI soit FAUX.

Une opération primitive doit participer à au moins une association par le rôle *est déclenché* et à au plus une association par le rôle *déclenche*.

5.2.2.2 Description des types d'association de la figure 5.2

a) Description

Les types d'association de la figure 5.2 modélisent les clauses de déclenchement de fin de sous-schéma par une boucle, une condition ou une opération primitive.

b) Entités associées

Etant donné que ces différents types d'association relèvent du même mécanisme, nous les décrirons globalement.

Les types d'association FIN_SCHEMA_BOUC, FIN_SCHEMA_COND et FIN_SCHEMA_OP associent le type d'entité SOUS_SCHEMA et respectivement, le type d'entité BOUCLE, CONDITION et OPERATION_PRIMITIVE.

Rôles : - "*est terminé par*" pour SOUS_SCHEMA.
- "*déclenche*" pour BOUCLE, CONDITION et OPERATION_PRIMITIVE.

Connectivités : - 0-N pour SOUS_SCHEMA.
- 0-1 pour BOUCLE et OPERATION_PRIMITIVE.
- 0-2 pour CONDITION.

c) Attributs

Seul l'association FIN_SCHEMA_COND a un attribut. En voici, la description :

- EVALUATION
  - l'évaluation d'une condition donnant un résultat booléen, l'attribut indique si la fin du sous-schéma, dans lequel se trouve la condition, est déclenchée dans le cas d'un résulat vrai ou faux.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant : {VRAI, FAUX}.

d) Contraintes d'intégrité

Un sous-schéma doit participer à au moins une association par le rôle *est terminé par*.

Si l'on regroupe les schémas 5.1 et 5.2, une boucle doit participer à deux et seulement deux associations par le rôle *déclenche*.

Si l'on regroupe les schémas 5.1 et 5.2, une condition doit participer à deux et seulement deux associations par le rôle *déclenche*. Ces deux associations doivent avoir des valeurs différentes pour leur attribut EVALUATION : ces valeurs sont soit VRAI soit FAUX.

Si l'on regroupe les schémas 5.1 et 5.2, une opération primitive doit participer à une et une seule association par le rôle *déclenche*.

Une boucle, une condition ou une opération primitive ne peut déclencher la fin d'un sous-schéma que si elle *fait partie* de ce sous-schéma. Nous dirons qu'une boucle, une condition ou une opération primitive *fait partie* d'un sous-schéma si elle a été déclenchée par ce sous-schéma ou si les concepts qui la déclenchent *font partie* eux-mêmes de ce sous-schéma.

5.2.3 Spécialisation des types d'entité boucle, condition et opération primitive

A partir de la description des types de composant boucle, condition et opération primitive donnée au chapitre 4, nous faisons les constatations suivantes :

- pour le type de composant condition, plusieurs clauses sont possibles (cfr. le point 4.4.2 la condition) et chacune d'elle travaille sur de l'information différente.
  Par exemple, la première clause demande le nom d'un attribut, l'identifiant d'un objet informationnel, un opérateur de comparaison

et une constante alpha-numérique. Tandis que la seconde clause demande l'identifiant d'un objet informationnel et l'identifiant d'un O.I.S.

- pour le type de composant boucle, plusieurs critères de sélection sont possibles (cfr. le point 4.4.3 : la boucle) et chacun d'eux travaille sur de l'information différente.
  Par exemple, le premier critère demande l'identifiant d'un O.I.S. tandis que le second critère demande une classe d'objet informationnel (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER PILE).
- pour le type de composant opération primitive, il existe plusieurs types d'opération (création, modification,...) qui demandent chacun leur information bien spécifique.

Nous procédons alors à une spécialisation des types d'entité en sous-types. La spécialisation constitue l'opération inverse de la généralisation. Elle est utilisée lorsque pour une application, l'information d'un type d'entité n'est plus suffisante et demande une information plus détaillée : on la le lui ajoute via une spécialisation du type d'entité en sous-types. Dans le type d'entité initial se trouve l'information générale, commune à tous les sous-types et chacun de ceux-ci rassemble une information plus détaillée qui lui est propre.

La décomposition en sous-types est exposé au point 5.2.4, pour les trois types de composant : condition, boucle et opération primitive. De cette explication suivra aux point 5.3, la définition et la description de chaque type d'entité.

5.2.4 Description des sous-types d'entité

5.2.4.1 La boucle

a) Description

En se référant au point 4.4.3, on voit que dans l'énoncé d'une boucle, nous devons travailler sur une information différente selon le type de critère choisi.

C'est pourquoi, il est possible de procéder à une spécialisation du type d'entité BOUCLE en sous-types. Nous obtenons ainsi deux types d'entité supplémentaires qui contiennent chacun l'information propre à un critère. Ces deux types d'entité se nomment : CRITERE_CONSTITUANTS et CRITERE_CLASSEMENT.

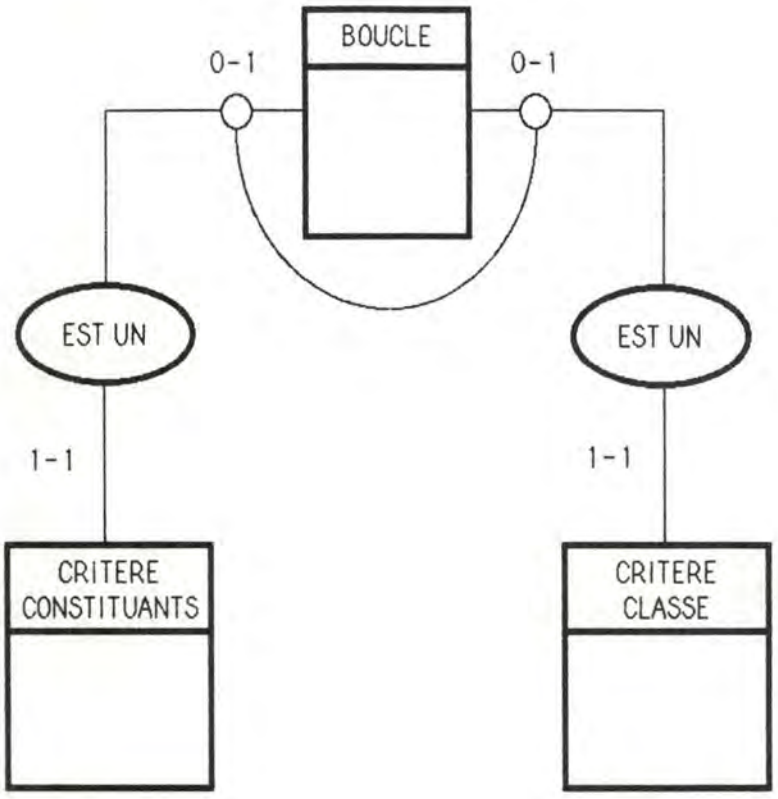


Fig. 5.3 : Spécialisation du type d'entité BOUCLE en sous-types

b) Explications et contraintes du schéma de la figure 5.3

- le type d'entité BOUCLE contient l'information commune à tous les critères. Cette information est composée, par exemple, du nom de la boucle (la liste complète des attributs de chaque entité est donnée au point 5.3).
- le type d'entité CRITERE_CONSTITUANTS contient l'information relative au critère qui sélectionne tous les composants d'un O.I.S. donné.
- le type d'entité CRITERE_CLASSEMENT contient l'information relative au critère qui sélectionne les occurrences de type INFORMATION d'une classe donnée.
- pour ce qui concerne le troisième critère, sélection de toutes les occurrences de type INFORMATION, la création d'un sous-type n'a pas été nécessaire pour la simple raison qu'il ne demande aucune information spécifique. Par conséquent, pour une occurrence donnée de type BOUCLE, celle-ci est associée au troisième critère, si elle n'est pas associée aux deux autres critères.

- contrainte 1 :

  Une boucle ne peut être associée qu'à un seul critère. Sur le schéma, cette contrainte s'exprime de la façon suivante : toute occurrence de type BOUCLE, non associée au troisième critère, ne

peut être associée qu'à au plus une occurrence du type CRITERE_CONSTITUANTS ou (exclusif) du type CRITERE_CLASSEMENT.

- contrainte 2 :

  Un critère ne peut être associé qu'à une seule boucle. Sur le schéma cette contrainte s'exprime de la façon suivante : toute occurrence de type CRITERE_CONSTITUANTS ou de type CRITERE_CLASSEMENT doit être associée à une et une seule occurrence de type BOUCLE.

Puisque les types d'entité CRITERE_CONSTITUANTS et CRITERE_CLASSEMENT sont des sous-types de BOUCLE, on peut encore dire qu'ils :

- héritent de l'identifiant de l'entité BOUCLE ainsi que de tous ses attributs,
- forment avec les attributs de l'entité BOUCLE, l'ensemble des informations décrivant une boucle.

5.2.4.2 La condition

a) Description

En se référant au point 4.4.2, on voit que dans l'énoncé d'une condition, nous devons travailler sur une information différente selon le type de clause choisi.

C'est pourquoi, il a été possible de procéder à une spécialisation du type d'entité CONDITION en sous-types. Nous obtenons ainsi trois types d'entité supplémentaires qui contiennent chacun l'information propre à une clause. Ces trois types d'entité se nomment : CLAUSE_ATTRIBUT, CLAUSE_APPARTENANCE et CLAUSE_PARAMETRE.

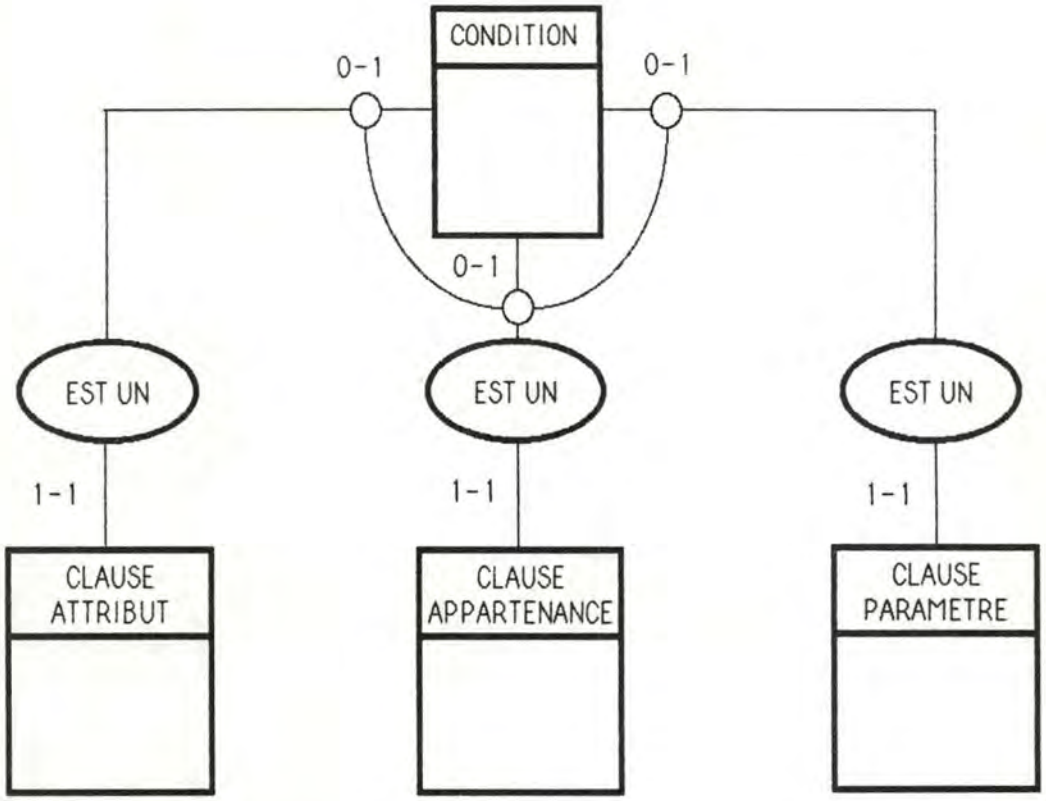


Fig 5.4 : Spécialisation du type d'entité CONDITION en sous-types

b) Explications et contraintes du schéma de la figure 5.4

- le type d'entité CONDITION contient l'information commune à toutes les clauses. Cette information est composée, par exemple, du nom de la condition (la liste complète des attributs de chaque entité est donnée au point 5.3).
- le type d'entité CLAUSE_ATTRIBUT contient l'information relative à la clause qui teste une valeur d'attribut pour une occurrence de type INFORMATION.
- le type d'entité CLAUSE_APPARTENANCE contient l'information relative à la clause qui teste l'inclusion d'un objet informationnel dans un O.I.S.
- le type d'entité CLAUSE_PARAMETRE contient l'information relative à la clause qui teste la valeur d'un paramètre.

- contrainte 1 :

  Une condition ne peut être associée qu'à une seule clause. Sur le schéma cette contrainte s'exprime de la façon suivante : toute occurrence de type CONDITION ne peut être associée qu'à au plus une occurrence de type CLAUSE_ATTRIBUT ou (exclusif) de type CLAUSE_APPARTENANCE ou (exclusif) de type CLAUSE_PARAMETRE.

- contrainte 2 :

  Une clause ne peut être associée qu'à une seule condition. Sur le schéma cette contrainte s'exprime de la façon suivante : toute occurrence de type CLAUSE_ATTRIBUT, de type CLAUSE_ APPARTENANCE ou de type CLAUSE_PARAMETRE doit être associée à une et une seule occurrence de type CONDITION.

Puisque les types d'entité CLAUSE-ATTRIBUT, CLAUSE-APPARTENACE et CLAUSE-PARAMETRE sont des sous-types de CONDITION, on peut encore dire qu'ils :

- héritent de l'identifiant de l'entité CONDITION ainsi que de tous ses attributs,
- forment avec les attributs de l'entité CONDITION, l'ensemble des informations décrivant une condition.

5.2.4.3 L'opération primitive

a) Description

En se référant au point 4.3.2, on voit que dans l'énoncé d'une opération primitive, nous devons travailler sur une information différente selon l'opération choisie.

C'est pourquoi, il a été possible de procéder à une spécialisation du type d'entité OPERATION_PRIMITIVE en sous-types. Nous obtenons ainsi un type d'entité supplémentaire pour chaque opération et qui contient l'information propre à une opération. Ces entités ont pour nom celui de l'opération qu'elle représente; nous aurons ainsi l'entité OPERATION_CREER, l'entité OPERATION_ MODIFIER, ...

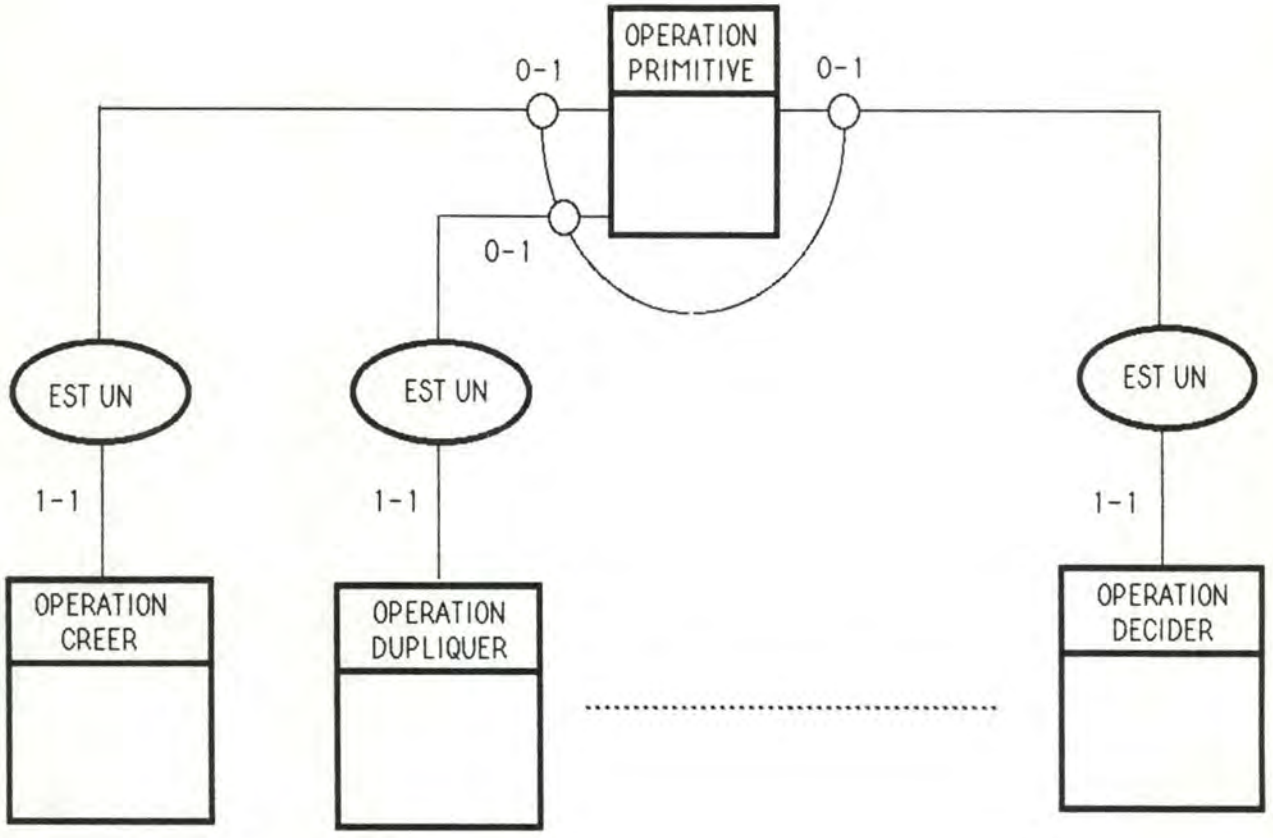


Fig 5.5 : Spécialisation du type d'entité OPERATION_PRIMITIVE en sous-types

b) Explications et contraintes du schéma de la figure 5.5

- le type d'entité OPERATION_PRIMITIVE contient l'information commune à toutes les opérations. Cette information est composée, par exemple, du nom de l'opération (la liste complète des attributs de chaque entité est donnée au point 5.3).
- chaque entité contient l'information spécifique de l'opération qu'elle représente.

- contrainte 1 :

  Une occurrence de type OPERATION_PRIMITIVE ne peut être associée qu'à une seule opération. Sur le schéma cette contrainte s'exprime de la façon suivante : toute occurrence de type OPERATION_PRIMITIVE ne peut être associée qu'à au plus une occurrence de type OPERATION_*nom_opération*

- contrainte 2 :

  Une opération ne peut être associée qu'à une seule occurrence de type OPERATION_PRIMITIVE. Sur le schéma cette contrainte s'exprime de la façon suivante : toute occurrence de type OPERATION_*nom_opération* doit être associée à une et une seule occurrence de type OPERATION_PRIMITIVE.

Puisque les types d'entité OPERATION_*nom_opération* sont des sous-types de OPERATION_PRIMITIVE, on peut encore dire qu'ils :

- héritent de l'identifiant de l'entité OPERATION_PRIMITIVE ainsi que de tous ses attributs,
- forment avec les attributs de l'entité OPERATION_PRIMITIVE, l'ensemble des informations décrivant une opération.

5.3 Modélisation de l'information contenue dans les phrases

5.3.1 Introduction

Cette deuxième étape donne une représentation de l'information propre à chaque type de composant (tâche, sous-schéma, boucle, condition, opération primitive). Cela se fera par le concept d'attribut du modèle E/A. L'idée qui sera adoptée ici est la suivante. Pour chaque composant, le système doit être capable de *mémoriser* et de *retrouver* l'information introduite par l'analyste. Pour ce faire, chaque élément d'information qui est susceptible d'être introduit, pour le composant, sera modélisé par un attribut. Ainsi, par exemple, dans la description de la tâche au point 4.5.4 A, nous avions :

TACHE nom_tâche
[DESCRIPTION : texte]
DECLENCHE SOUS_SCHEMA nom_sous-schéma

Avec cette phrase, l'analyste doit introduire un nom de tâche, il peut introduire un texte décrivant cette tâche et il doit identifier le sous-schéma qui sera déclenché.

La notion de déclenchement étant représenté par une association, dans l'exemple ci-dessus, par une association de déclenchement entre la tâche et le sous-schéma, il reste à représenter dans la base de données du langage le nom de la tâche et sa description. Pour cela, dans le type d'entité TACHE, nous aurons :

- un attribut pour recevoir le nom de la tâche,
- un attribut pour recevoir le texte de la description de la tâche.

5.3.2 Description des types d'entité

En reprenant les schémas des figures 5.1, 5.2, 5.3, 5.4 et 5.5, nous pourrions les remettre tous sur un seul schéma. Dans ce schéma, nous aurions un ensemble de types d'entité et de types d'association. Ce paragraphe va décrire ces types d'entité plus précisément.

Pour chaque type d'entité, nous décrivons les attributs qui le composent; pour chacun d'eux nous donnons sa définition, son type et son domaine de valeurs.

1) TACHE (cfr. le point 4.5.4 A)

- NOM_TACHE
  - identifiant de la tâche qui est simulée.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères

commençant par une lettre.

- DESCRIPTION
  - texte décrivant la tâche sous forme de commentaire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères.

2) SOUS_SCHEMA (cfr. le point 4.5.4 B)

- NOM_SOUS_SCHEMA
  - identifiant du sous-schéma.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- DESCRIPTION
  - texte décrivant le sous-schéma sous forme de commentaire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères.

3) BOUCLE (cfr. le point 4.5.3 C)

- NOM_BOUCLE
  - identifiant de la boucle.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- DESCRIPTION
  - texte décrivant la tâche sous forme de commentaire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères.

- NOM_VARIABLE
  - une boucle traite un ensemble d'articles. A chaque passage dans la boucle, l'identifiant d'un des articles sélectionné est affecté à la variable de la boucle. Cet attribut donne le nom de cette variable.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

4) CRITERE_CONSTITUANT (cfr. le point 4.4.3 A)

- CLASSE
  - spécifie la classe de l'O.I.S. dont on veut trouver les constituants.
  - attribut de type élémentaire, obligatoire et simple.

- le domaine de valeurs est l'ensemble suivant : {DOSSIER, FICHIER, PILE}

- TYPE
  - spécifie le type de l'O.I.S. dont on veut trouver les constituants.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - spécifie le nom de l'O.I.S. dont on veut trouver les constituants.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- DESIR
  - indique le critère de sélection des constituants dans l'O.I.S.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble {OIE, OIS, TOUS}.

- NOM_VARIABLE
  - nom de variable qui référence l'O.I.S. dont on veut rechercher les constituants.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des noms de variable qui existent dans le système.

5) CRITERE_CLASSE (cfr. le point 4.4.3 B)

- CLASSE
  - spécifie la classe de l'objet informationnel dont on veut retrouver toutes les occurrences.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs l'ensemble suivant : {MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE}.

6) CONDITION (cfr. le point 4.5.4 D)

- NOM_CONDITION
  - identifiant de la condition.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- DESCRIPTION
  - texte décrivant la condition sous forme de commentaire.

- attribut élémentaire, facultatif et simple.
- le domaine de valeurs est l'ensemble des chaînes de caractères.

7) CLAUSE_ATTRIBUT (cfr. le point 4.4.2 A)

- NOM_ATTRIBUT
  - spécifie le nom de l'attribut dont la valeur est testée.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant : {CLASSE, TYPE, NOM, NUM_COPIE, ETAT, ECHEANCE, ORDRE_CLAS, ETAT_CONTENU}.

- CST_ALPHA_NUMERIQUE
  - spécifie la valeur à laquelle est comparée la valeur de l'attribut dont le nom se trouve dans NOM_ATTRIBUT.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères.

- OPERATEUR
  - il spécifie l'opérateur de comparaison entre la valeur de l'attribut, dont le nom se trouve dans NOM_ATTRIBUT, et la valeur de l'attribut CST_ALPHA_NUMERIQUE.
  - attribut élémentaire, obligatoire et simple.
  - domaine de valeurs est l'ensemble suivant : {=, !=, >, <, >=, <=}.

- NOM_VARIABLE
  - spécifie le nom de la variable qui référence l'occurrence de type INFORMATION sur laquelle porte la condition.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des noms de variable qui existent dans le système.

- CLASSE
  - spécifie la classe de l'occurrence de type INFORMATION dont l'attribut est testé.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE}.

- TYPE
  - spécifie le type de l'objet informationnel dont l'attribut est testé.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - spécifie le nom de l'objet informationnel dont l'attribut est testé.

- attribut élémentaire, facultatif et simple.
- le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - spécifie le numéro de la copie de l'objet informationnel dont l'attribut est testé.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des nombres entiers.

8) CLAUSE APPARTENANCE (voir le point 4.4.2 B)

- OI_NOM_VARIABLE
  - spécifie le nom de la variable qui référence l'objet informationnel qui est traité dans la condition.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des noms de variable qui existent dans le système.

- OI_CLASSE
  - spécifie la classe de l'objet informationnel dont l'appartenance à un O.I.S. est testée.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE).

- OI_TYPE
  - spécifie le type de l'objet informationnel dont l'appartenance à un O.I.S. est testée.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- OI_NOM
  - spécifie le nom de l'objet informationnel dont l'appartenance à un O.I.S. est testée.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- OI_NUM_COPIE
  - spécifie le numéro de copie de l'objet informationnel dont l'appartenance à un O.I.S. est testée.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des nombres entiers.

- OIS_NOM_VARIABLE
  - spécifie le nom de variable qui référence l'objet informationnel structurant.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des noms de variable qui existent dans le système.

- OIS_CLASSE
  - spécifie la classe de l'O.I.S.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {DOSSIER, FICHIER, PILE}.

- OIS_TYPE
  - spécifie le type de l'O.I.S.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- OIS_NOM
  - spécifie le nom de l'O.I.S.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

9) CLAUSE_PARAMETRE (voir le point 4.4.2 C)

- NOM_PARAMETRE
  - spécifie le nom du paramètre dont la valeur est testée.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des paramètres qui sont définis dans le système.

- OPERATEUR
  - spécifie l'opérateur de comparaison entre la valeur du paramètre et une constante alpha-numérique.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant : {=, >, <, >=, <=, !=}.

- CST_ALPHA_NUMERIQUE
  - spécifie la valeur de comparaison du paramètre.
  - attribut élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères.

10) Entité OPERATION PRIMITIVE (cfr. le point 4.5.3 E)

- NOM_OPERATION
  - identifiant de l'opération primitive dans la dynamique.
  - attribut de type élementaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- DESCRIPTION
  - texte décrivant l'opération primitive sous forme de commentaire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

11) Entité OPERATION CREER (cfr. le point 4.3.3 A)

- CLASSE
  - classe de l'objet informationnel à créer.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant: {MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE}.

- TYPE
  - type de l'objet informationnel à créer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à créer.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ETAT
  - état de l'objet informationnel à créer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à créer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

- ORDRE_CLAS
  - ordre de classement de l'objet informationnel à créer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (LIFO, FIFO, ALPHABETIQUE, ALPHABETIQUE INVERSE,AUCUN).

12) Entité OPERATION REPRODUIRE (cfr. le point 4.3.3 B)

- NOM_VARIABLE
  - nom de la variable qui référence l'O.I.E. à reproduire.
  - attribut de type élémentaire,facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à reproduire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT).

- TYPE
  - type de l'objet informationnel à reproduire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à reproduire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NBR_COPIE
  - nombre d'exemplaires qui doivent être reproduits par l'opération.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- NUM_DEPART
  - numéro de départ à partir duquel les copies doivent être numérotées.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

13) Entité OPERATION RECEVOIR (cfr. le point 4.3.3 C)

- MODE_RECEPTION
  - mode dans lequel doit s'effectuer la reception de l'objet

informationnel.
- attribut de type élémentaire, obligatoire et simple.
- le domaine de valeurs est l'ensemble suivant : (ECRIT,ORAL).

- CLASSE
  - classe de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER).

- TYPE
  - type de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

- ORDRE_CLAS
  - ordre de classement de l'objet informationnel à recevoir.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (LIFO, FIFO, ALPHABETIQUE, ALPHABETIQUE INVERSE, AUCUN).

- COMPOSANT:
  - dans le cas de la réception d'un O.I.S., spécifie quels en sont les composants.
  - attribut de type décomposable, facultatif et répétitif.
  - l'attribut est décompsable en six attributs :
    - ◇ COMPOSANT.CLASSE
      - indique la classe du composant.
      - attribut de type élémentaire, obligatoire et simple.
      - le domaine de valeurs est l'ensemble suivant : {MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER}.
    - ◇ COMPOSANT.TYPE
      - indique le type du composant.
      - attribut de type élémentaire, facultatif et simple.
      - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.
    - ◇ COMPOSANT.NOM
      - indique le nom du composant.
      - attribut de type élémentaire, obligatoire et simple.
      - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.
    - ◇ COMPOSANT.NUM_COPIE
      - indique le numéro de copie du composant.
      - attribut de type élémentaire, facultatif et simple.
      - le domaine de valeurs est celui des nombres entiers.
    - ◇ COMPOSANT.ETAT
      - indique l'état du composant.
      - attribut de type élémentaire, facultatif et simple.
      - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.
    - ◇ COMPOSOSANT.ECHEANCE
      - indique l'échéance du composant.
      - attribut de type élémentaire, facultatif et simple.
      - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

14) Entité OPERATION COMMUNIQUER (cfr. le point 4.3.3 D)

- MODE_COMMUNICATION
  - mode dans lequel doit s'effectuer la communication de l'objet informationnel.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant : {ECRIT, ORAL}.

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à communiquer.

- attribut de type élémentaire,facultatif et simple.
- le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à communiquer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER).

- TYPE
  - type de l'objet informationnel à communiquer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à communiquer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à communiquer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état de l'objet informationnel à communiquer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à communiquer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

15) Entité OPERATION CONSULTER (cfr. le point 4.3.3 E)

- NOM_VARIABLE
  - nom de la variable qui référence l'O.I.E. à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {MESSAGE, FORMULAIRE, DOCUMENT}.

- TYPE
  - type de l'objet informationnel à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état de l'objet informationnel à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à consulter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

16) Entité OPERATION FEUILLETER (cfr. le point 4.3.3 F)

- NOM_VARIABLE
  - nom de la variable qui référence l'O.I.S. à feuilleter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à feuilleter.
  - attribut de type élémentaire, facultatif et simple.

- le domaine de valeurs est l'ensemble suivant : {DOSSIER, FICHIER, PILE}.

- TYPE
  - type de l'objet informationnel à feuilleter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à feuilleter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ETAT
  - état de l'objet informationnel à feuilleter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à feuilleter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

- ORDRE_FEUILLETAGE
  - ordre dans lequel les constituants de l'O.I.S. spécifié doivent être feuilletés.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {LIFO, FIFO, ALPHABETIQUE, ALPHABETIQUE INVERSE}.

17) Entité OPERATION ENLEVER (cfr. le point 4.3.3 G)

- NOM_VARIABLE
  - nom de la variable qui référence la pile dont on doit enlever le premier constituant.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM_PILE
  - nom de la pile dont on doit enlever le premier constituant.
  - attribut de type élémentaire, facultatif et simple.

- le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- VARIABLE_CONSTITUANT
  - nom de la variable qui référence l'objet informationnel que l'on a enlevé de la pile.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

18) Entité OPERATION MODIFIER (cfr. le point 4.3.3 H)

- NOM_VARIABLE
  - nom de la variable qui référence le document à modifier.
  - attribut de type élémentaire,facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- TYPE
  - type de document à modifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom du document à modifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie du document à modifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état du document à modifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance du document à modifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

- NOM_MESSAGE
  - nom du message brouillon à partir duquel se fera la modification du document spécifié.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE_MESSAGE
  - numéro de copie du message brouillon à partir duquel se fera la modification du document spécifié.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

19) Entité OPERATION COMPLETER (cfr. le point 4.3.3 l)

- NOM_VARIABLE
  - nom de la variable qui référence le formulaire à compléter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- TYPE
  - type de formulaire à compléter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom du formulaire à compléter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie du formulaire à compléter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état du formulaire à compléter.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance du formulaire à compléter.

- attribut de type élémentaire, facultatif et simple.
- le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

20) Entité OPERATION VERIFIER (cfr. le point 4.3.3 J)

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {FORMULAIRE, DOSSIER}.

- TYPE
  - type de l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état de l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à vérifier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

- NOM_PARAMETRE
  - nom de la variable qui contiendra la réponse entrée par l'utilisateur lors de cette opération de vérification.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

21) Entité OPERATION DETRUIRE (cfr. le point 4.3.3 K)

- NOM_VARIABLE
  - nom de la variable qui référence l'O.I.E. à détruire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à détruire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT).

- TYPE
  - type de l'objet informationnel à détruire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à détruire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à détruire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état de l'objet informationnel à détruire.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à détruire.

- attribut de type élémentaire, facultatif et simple.
- le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

22) Entité OPERATION ARCHIVER (cfr. le point 4.3.3 L)

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {FORMULAIRE, DOCUMENT, DOSSIER, FICHIER}.

- TYPE
  - type de l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- ETAT
  - état de l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à archiver.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

23) Entité OPERATION TRIER (cfr. le point 4.3.3 M)

- NOM_VARIABLE
  - nom de la variable qui référence l'O.I.S. à trier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à trier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {DOSSIER, FICHIER, PILE}.

- TYPE
  - type de l'objet informationnel à trier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à trier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ETAT
  - état de l'objet informationnel à trier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- ECHEANCE
  - échéance de l'objet informationnel à trier.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaines de caractères représentant une date sous le format AAAA/MM/JJ.

- ORDRE_TRI
  - ordre dans lequel les constituants de l'O.I.S. spécifié doivent être triés.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {LIFO, FIFO, ALPHABETIQUE, ALPHABETIQUE INVERSE}.

24) Entité OPERATION CLASSER (cfr. le point 4.3.3 N)

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à classer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à classer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : {MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER}.

- TYPE
  - type de l'objet informationnel à classer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à classer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à classer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- DEST_CLASSE
  - classe de l'objet de informationnel dans lequel on doit classer l'objet informationnel spécifié.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant : {DOSSIER, FICHIER, PILE}.

- DEST_TYPE
  - type de l'objet de informationnel dans lequel on doit classer l'objet informationnel spécifié.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- DEST_NOM
  - nom de l'objet de informationnel dans lequel on doit classer l'objet informationnel spécifié.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

25) Entité OPERATION RANGER (cfr. le point 4.3.3 O)

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à ranger.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à ranger.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE).

- TYPE
  - type de l'objet informationnel à ranger.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à ranger.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à ranger.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- DEST_CLASSE
  - classe de l'objet de rangement dans lequel on doit ranger l'objet informationnel spécifié.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble suivant : (ETAGERE, BOITE, TIROIR, FARDE).

- DEST_NOM
  - nom de l'objet de rangement dans lequel on doit ranger l'objet informationnel spécifié.
  - attribut de type élémentaire, obligatoire et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

26) Entité OPERATION REPLACER (cfr. le point 4.3.3 P)

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à replacer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à replacer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE).

- TYPE
  - type de l'objet informationnel à replacer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à replacer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à replacer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- GR_CLASSE
  - classe du groupe des objets informationnels à replacer.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (MESSAGE, FORMULAIRE, DOCUMENT, DOSSIER, FICHIER, PILE).

- GR_TYPE
  - type du groupe des objet informationnel à replacer.

- attribut de type élémentaire, facultatif et simple.
- le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

27) Entité OPERATION DECIDER (cfr. le point 4.3.3 Q)

- NOM_VARIABLE
  - nom de la variable qui référence l'objet informationnel à partir duquel la décision doit être prise.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- CLASSE
  - classe de l'objet informationnel à partir duquel la décision doit être prise.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble suivant : (FORMULAIRE, DOCUMENT, DOSSIER, FICHIER).

- TYPE
  - type de l'objet informationnel à partir duquel la décision doit être prise.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NOM
  - nom de l'objet informationnel à partir duquel la décision doit être prise.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

- NUM_COPIE
  - numéro de copie de l'objet informationnel à partir duquel la décision doit être prise.
  - attribut de type élémentaire, facultatif et simple.
  - le domaine de valeurs est celui des nombres entiers.

- NOM_PARAMETRE
  - nom de la variable qui contienté la réponse entrée par l'utilisateur lors de cette opération de prise de décision.
  - attribut élémentaire, facultatif et simple.
  - le domaine de valeurs est l'ensemble des chaînes de caractères commençant par une lettre.

5.4 Les contrôles

Tout au long de la simulation de la tâche, toute une série de contrôles devra être réalisée pour vérifier l'exactitude des informations manipulées. Le moment où les contrôles sont exécutés peut constituer un critère de regroupement de ceux-ci. Ainsi, nous obtenons trois groupes de contrôles :

- les contrôles réalisés sur les spécifications introduites par l'analyste,
- les contrôles réalisés lors de la mise à jour de la base de données,
- les contrôles réalisés avant l'exécution d'un composant.

5.4.1 Vérification des spécifications de l'analyste

Lorsque l'analyste a spécifié la tâche qu'il désire simuler, il faut que le système vérifie si les spécifications introduites respectent la syntaxe et la sémantique du modèle.

Une bonne spécification est une spécification [BOPI 83] :

- complète : il n'existe pas de composants référencés mais non complètement décrits.
- cohérente : il n'existe pas de contradiction dans la description.

Nous décrivons dans ce point les contrôles de complétude et de cohérence pour chacune des opérations primitives, pour chacune des structures d'enchaînement, pour la construction de la tâche et enfin pour la déclaration de l'environnement.

Tant que les spécifications introduites ne respecte pas ces contrôles, l'animation graphique de la tâche ne pourra pas commencer et l'analyste devra apporter les corrections nécessaires à ses spécifications.

A) Les contrôles de complétude

Une spécification sera dite *incomplète* si par rapport au modèle de spécification, des clauses qui devraient caractériser un composant ne figurent pas dans la description de celui-ci.

1) Complétude des opérations primitives

La description d'une opération primitive sera complète si l'analyste a spécifié au moins toutes les clauses obligatoires et que pour chacune d'elles, il a respecté la syntaxe établie au point 4.3.

2) Complétude des structures d'enchaînement

La description d'une structure d'enchaînement sera complète si l'analyste a spécifié au moins toutes les clauses obligatoires et que pour chacune d'elles , il a respecté la syntaxe établie au point 4.4.

3) Complétude de la construction de la tâche

La description d'une tâche ou d'un sous-schéma sera complète si l'analyste a spécifié au moins toutes les clauses obligatoires et que pour chacune d'elles, il a respecté la syntaxe établie au point 4.5.

4) Complétude de la déclaration de l'environnement

La description d'un objet ou d'une association entre objets sera complète si l'analyste a spécifié au moins toutes les clauses obligatoires et que pour chacune d'elles, il a respecté la syntaxe établie au point 4.6.

De plus, le système vérifiera :

- que l'analyste a bien spécifié un objet bureau,
- que si l'analyste a créé une boîte IN et/ou une boîte OUT, alors il leur a associé une pile.

B) Les contrôles de cohérence

Une spécification sera dite *non cohérente* si elle présente des contradictions ou des incompatibilités vis-à-vis de la dynamique définie dans le modèle ou vis-à-vis de ce qui a déjà été décrit.

1) La cohérence des opérations primitives (cfr. 4.3)

La syntaxe des opérations primitives telle qu'elle a été établie au point 4.3. rend impossible les erreurs de cohérence.

L'emploi de variables dans les opérations primitives est tout à fait faisable à la condition de vérifier que cette variable a été déclarée soit dans une *boucle* soit dans une opération primitive *enlever*.

2) La cohérence des structures d'enchaînement (cfr. 4.4)

- La condition

a) Contrôle de la cohérence pour la clause *test de la valeur d'un attribut*

- si on teste un attribut d'un O.I.S., alors cet attribut ne peut être le numéro de copie (NUM_COPIE).
- si on teste un attribut d'un O.I.E., alors cet attribut ne peut être ni l'ordre de classement (ORDRE_CLAS) ni l'état du contenu (ETAT_CONTENU).
- si on teste un attribut d'un message, d'un fichier ou d'une pile, alors cet attribut ne peut être le type (TYPE).
- si on utilise une variable, il faut veiller à ce que celle-ci ait été déclarée soit dans une *boucle* ou soit dans une opération primitive *enlever*.

b) Contrôle de cohérence pour la clause *test de l'appartenance d'un O.I.*

- si on teste l'appartenance d'une variable dans une autre variable, elles ne peuvent pas avoir le même nom.
- si on utilise une variable, il faut veiller à ce que celle-ci ait été déclarée soit dans une *boucle* soit dans une opération primitive *enlever*.

c) Contrôle de cohérence pour la clause *test de la valeur d'un paramètre*

- lors de l'utilisation de cette clause, il faut veiller à ce que le paramètre utilisé ait été déclaré soit dans une opération primitive *vérifier* soit dans une opération primitive *décider*.

- La boucle

  La syntaxe employée pour la description des critères (cfr. 4.4.3) rend impossible les erreurs de cohérence.

3) Cohérence de la construction de la tâche (cfr. 4.5)

Nous décrivons maintenant l'ensemble des contrôles à effectuer sur chacun des types de composant : tâche, sous-schéma, boucle, condition et opération primitive.

- pour chacun des composants, le nom doit être identifiant.
- pour toutes les clauses DECLENCHE *x*, le nom *x* identifie un composant existant.
- pour toutes les clauses DECLENCHEE PAR *x*, le nom *x* identifie un composant existant.
- tout composant faisant partie d'un sous-schéma ne peut pas déclencher la fin d'un autre sous-schéma.
- tout sous-schéma doit être terminé par au moins un composant.
- il doit y avoir correspondance entre les composants déclenchés :

Composant 1	Composant 2
DECLENCHE Composant 2	DECLENCHEE PAR Composant 1
...	...

4) Cohérence de la déclaration de l'environnement

a) Cohérence des objets créés

- Pour la déclaration d'un objet informationnel :
  - l'objet qui est crée ne peut pas déjà exister,
  - si l'objet créé est un O.I.E., alors la clause AVEC ORDRE DE CLASSEMENT doit être omise.
  - si l'objet créé est une pile alors l'ordre de classement doit être *lifo*.

- Pour la création d'un objet de rangement :
  - l'objet qui est créé ne peut pas déjà exister,

- Pour la création d'un outil de travail ou d'un objet d'interface :
  - un et un seul objet peut être créé (par exemple, il ne peut y avoir qu'un seul téléphone)

- Une table de travail doit avoir été déclarée.

b) Cohérence des associations créées entre les objets

- Pour la création d'une association de rangement, l'objet de rangement et l'objet informationnel doivent exister.

- Pour la création d'une association de classement, les deux objets informationnels doivent exister.

- Pour la création d'une association de composition :
  - tout tiroir doit se trouver dans une armoire,
  - toute farde doit se trouver soit dans une boîte, dans un tiroir ou dans une étagère,
  - toute boîte doit se trouver dans une étagère.

- Si une boîte IN ou une boîte OUT a été déclarée, alors il doit leur être associée une pile.

Un contrôle de cohérence supplémentaire à effectuer est de vérifier que tout objet informationnel créé doit participer à une et une seule des trois associations suivantes :

- une association de classement,
- une association d'interface,
- une association de rangement.

5.4.2 Contrôles effectués lors de la mise à jour de la base de données

Les contrôles effectués lors de la mise à jour de la base de données servent à vérifier que la base de données reste dans un état cohérent. La base de données sera dans un état cohérent si toutes les contraintes d'intégrité qui ont été définies sur elle sont respectées.

5.4.3 Contrôles effectués avant l'exécution d'un composant

Lorsque le système est prêt à exécuter un composant, il faut encore vérifier que les objets manipulés par ce composant existent dans le système. Ces contrôles ne peuvent s'exécuter à un autre moment car la base de données est continuellement mise à jour lors de la simulation de la tâche. De ce fait l'information manipulée par un composant n'est disponible que juste avant son exécution.

5.5 Conclusion

La définition du langage telle qu'elle a été faite au chapitre 4, rend la représentation de celui-ci dans une base de données très facile.

La notion de *type* et d'*occurrence* étant présente dans cette définition avec les notions de *type de composant* et de *composant*, leur représentation peut se faire au moyen de types d'entité. De plus comme chaque composant possède un nom qui l'identifie, leur représentation en tant qu'occurrence est facile.

L'exécution de la tâche étant une suite de déclenchements entre composants, la mémorisation de l'ordre d'exécution de la tâche est facile grâce à ces relations de *déclenchement*.

La syntaxe de toutes les phrases étant établie, il est possible de déterminer, pour une phrase, quels sont les informations que l'analyste pourra introduire. Cette liste faite, il est alors possible de représenter cette information au moyen d'attributs.

Le chapitre suivant est destiné à présenter l'animation graphique des tâches de bureau; nous y verrons notamment quel type d'animation nous avons choisit.

Chapitre 6
Le graphisme

CHAPITRE 6 : LE GRAPHISME

L'emploi d'un outil d'animation graphique pour la simulation des tâches de bureau constitue l'**originalité** du mémoire et donc du projet dans lequel il s'insère. C'est pourquoi arrivés à ce stade de l'analyse, nous consacrons un chapitre au graphisme.

Nous commençons par une introduction qui explique quelles sont les raisons qui sont à la source de l'emploi du graphisme. La justification étant donnée, nous expliquons ensuite comment nous créons la simulation graphique d'une tâche de façon générale et par après la description graphique de chaque opération primitive. Pour terminer, nous décrivons l'information qu'il faut ajouter au système d'information pour pouvoir gérer le graphisme.

6.1 Introduction

L'informatique fait donc de plus en plus intrusion dans les environnements de bureau. Avec elle, toute une série de machines de très haute technologie sont proposées aux employés de bureau. Cette confrontation de l'homme avec la machine ne se fait pas sans poser d'innombrables problèmes sociaux et psychologiques dûs aux boulversements entraînés par ces machines dans le travail de bureau.

De plus l'emploi d'un ordinateur par une personne dont l'informatique n'est pas la spécialité n'est pas du tout chose facile. La plupart des logiciels proposés à ces utilisateurs ont généralement été conçus sans leur demander conseils et ensuite s'accompagnent d'un mode d'emploi lourd, mal fait et qui ne sert à rien sinon à décourager l'utilisateur.

Pour faire comprendre un logiciel à un utilisateur, ce n'est donc pas en écrivant de beaux manuels d'emploi après la réalisation du logiciel que l'on peut espérer y arriver. C'est au moment de la conception du logiciel et de ses interfaces qu'il faut y penser. La règle d'or dans ce domaine serait : "mieux l'interface est conçue et moins la documentation est nécessaire". [CLAN 83].

Pour améliorer les interfaces des logiciels toute une série de techniques ont été proposées en vue de mettre sur pied des systèmes *user-friendly*. En guise de parenthèse, W.J. Raduchel dans [RADU 84] donne la définition suivante à propos des systèmes user-friendly : "un système user-friendly aide à produire des solutions précises dans un temps plus court et à des coûts moins élevés que d'autres solutions alternatives." En effet des systèmes user-friendly ou conviviaux sont faciles à utiliser et leur apprentissage est aussi très simple.

Dans le domaine de l'interfaçage, la technique la plus intéressante est sans nul doute l'emploi du graphisme. Il permet, en effet, de reproduire (de dessiner) à l'écran des figures qui représentent des objets physiques du bureau; c'est ce

qu'on appellera des icônes. L'utilisateur se trouve alors en face d'un écran où sont dessinés tous les objets qu'il connait et par la sélection de l'un d'entre eux, il peut exécuter certains travaux : c'est ce qu'on appele la métaphore du bureau (*desktop metaphor*) [CLAN 83] et [WARF 83, article de G.M White].

Ce principe de la métaphore du bureau a été imaginée par le Dr. Alan Kay dans le cadre d'un principe plus général qui est la métaphore physique de l'ordinateur (*physical metaphor of computing*). C'est ainsi que Alan Kay peut être considéré comme le père intellectuel des systèmes Xérox Star ou Apple Lisa qui s'inspirent directement de ses théories. Pour lui, l'utilisateur a de meilleures intuitions, sur les possibilités qui lui sont offertes, en regardant des objets qu'il connait dessinés à l'écran. Ainsi, si l'utilisateur voit sur l'écran deux symboles qui représentent deux feuilles de papier se recouvrant partiellement, il pourra en déduire facilement qu'il doit être possible de mettre en avant plan la feuille qui se trouvait en arrière plan et cela sans connaître au préalable les possibilités réelles du système.

Notre mémoire s'inspire directement de ce principe en présentant à l'utilisateur l'exécution d'une de ses tâches sur des symboles (icônes) qui représentent des objets physiques de son bureau et qu'il connait très bien. Une telle simulation sera plus intuitive et plus compréhensible pour lui qu'un dossier d'un certain volume écrit par des informaticiens dans un langage pas toujours très compréhensible.

Le mémoire que nous réalisons, s'il s'inspire de la métaphore du bureau, va encore plus loin puisque non seulement nous recréons l'environnement de travail de l'utilisateur mais en plus, nous l'animons !

6.2 Description du type d'animation choisi

Nous exposons ici certains choix qui ont été faits du point de vue de l'animation graphique. Lors de l'évaluation du prototype, il s'agira d'examiner ces choix et de voir s'ils ont été pertinents ou non et éventuellement de les remettre en cause pour une évolution future du prototype.

Nous prenons l'option de nous limiter à une animation graphique en deux dimensions plutôt qu'une animation graphique en trois dimensions. Les raisons principales ont été d'une part le manque d'outil permettant une animation tri-dimensionnelle efficace et bon marché et d'autre part la compléxité de celle-ci.

Ce choix va avoir des conséquences tant au niveau de la représentation des objets que de l'animation des opérations primitives.

6.2.1 Représentation graphique des objets

Chaque type d'objet est représenté par une icône qui lui est propre (cfr. figure 6.1 à 6.4).

Fig. 6.1 : Icônes représentant les objets informationnels.

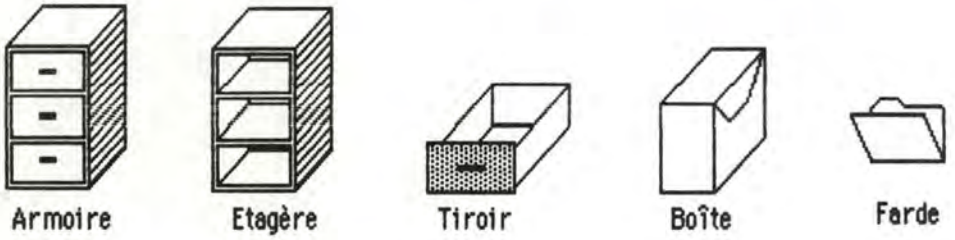


Fig. 6.2 : Icônes représentant les objets de rangement.

Fig. 6.3 : Icônes représentant les objets d'interface

Fig. 6.4 : Icônes représentant les outils de travail

6.2.2 Animation graphique des objets

Au niveau de l'animation graphique portant sur les objets, on peut la scinder en deux parties :

- la simulation du flux informationnel précédant et suivant l'opération primitive. Exemple : amener un formulaire sur la table de travail avant de le compléter. Cette partie sera simulée par un déplacement d'une icône représentant l'objet informationnel concerné.

- la simulation de l'opération primitive proprement dite qui est quant à elle simulée par un agrandissement de l'icône représentant l'objet informationnel et ensuite par une animation sur cet agrandissement. Exemple :

agrandissement d'un formulaire et simulation de son complètement champ par champ. Nous dénommons cette partie par le terme de *zoom*. Cette partie n'interviendra que lorsque l'objet informationnel sera sur la table de travail.

Certaines opérations, de par leur nature, ne nécessitera qu'une simulation du flux informationnel (exemple : destruction d'un formulaire) tandis que d'autres ne nécessitera qu'un agrandissement (exemple : consultation d'un formulaire qui se trouve déjà sur la table de travail).

6.3 Description graphique des opération primitives

Dans ce point, nous exposons de manière précise une opération primitive et ensuite, nous décrivons les autres de manière plus succinte.

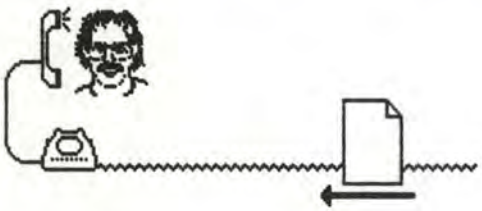
6.3.1 Description précise d'une opération primitive

Nous avons choisi d'exposer ici, de manière précise, l'opération de réception d'un message par le téléphone.

1ère étape : le téléphone sonne.

2ème étape : le téléphone est décroché.

3ème étape : un message apparaît et suit le fil du téléphone.



4ème étape : le message est déposé près du téléphone si l'utilisateur garde une trace écrite de la conversation.

5ème étape : le téléphone est raccroché.

6.3.2 Description succincte des opérations primitives

A) L'opération de **création**

- l'objet informationnel à créer apparaît à l'endroit de l'écran marqué par le symbole NEW et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

B) L'opération de **reproduction**

- l'objet informationnel à reproduire sort de son lieu de rangement ou de classement et se dirige vers la photocopieuse.
- les copies sortent une à une.
- l'original et les copies sont déposées sur la photocopieuse.

C) L'opération de **réception**

a) réception par la BOITE IN

- la porte s'ouvre, l'objet informationnel à recevoir apparaît, la porte se referme.
- l'objet informationnel se dirige vers la boîte IN.
- l'objet informationnel est déposé au sommet de la pile de la boîte IN.

b) réception d'un message par le téléphone

- le téléphone sonne et est décroché.
- un message apparaît au bout du fil du téléphone et se dirige vers celui-ci.
- le téléphone est raccroché et le message est déposé près du téléphone.

D) L'opération de **communication**

a) communication par la BOITE OUT

- l'objet informationnel à communiquer sort de son lieu de rangement ou de classement et se dirige vers la BOITE OUT.
- l'objet informationnel est déposé au sommet de la pile de la BOITE OUT

b) communication d'un message par le téléphone

- si une source écrite est spécifiée, alors celle-ci sort de son lieu de rangement ou de classement et se dirige vers le téléphone.
- le téléphone est décroché, un message apparaît près de celui-ci et s'éloigne en suivant le fil du téléphone.
- si une source écrite a été utilisée, celle-ci retourne dans son lieu de rangement ou de classement.
- le téléphone est raccroché.

E) L'opération de **consultation**

- l'objet informationnel à consulter sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. Un squelette dépendant du type d'objet informationnel consulté apparaît à l'écran de même que ses attributs.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

F) L'opération de **feuilletage**

- l'objet informationnel à feuilleter sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. La liste des constituants de l'objet informationnel à feuilleter apparaît à l'écran, de même que ses attributs.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

G) L'opération d'**enlèvement**

- on sort la pile concernée de son lieu de rangement ou de classement
- on enlève le premier élément de cette pile.
- la pile est remise à sa place.
- le premier élément se dirige vers la table de travail et est déposé sur celle-ci.

H) L'opération de **modification**

- l'objet informationnel à modifier sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- si un message brouillon est utilisé, alors celui-ci sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. Un squelette dépendant du type d'objet informationnel à modifier apparaît

à l'écran de même que ses attributs.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail, de même que le brouillon éventuel.

I) L'opération de **complètement**

- l'objet informationnel à compléter sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. Un squelette dépendant du type d'objet informationnel à compléter apparaît à l'écran de même que ses attributs.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

J) L'opération de **vérification**

- l'objet informationnel à vérifier sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. Un squelette dépendant du type d'objet informationnel à vérifier apparaît à l'écran de même que ses attributs.
- l'utilisateur entre sa réponse concernant la complétude ou non de l'objet informationnel à vérifier.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

K) L'opération de **destruction**

- l'objet informationnel à détruire sort de son lieu de rangement ou de classement et se dirige vers la poubelle.
- la poubelle s'ouvre, l'objet informationnel disparaît et la poubelle se referme.

L) L'opération d'**archivage**

- l'objet informationnel à archiver sort de son lieu de rangement ou de classement et se dirige vers la boîte d'archivage.
- l'objet informationnel est déposé dans la boîte d'archivage.

M) L'opération de **tri**

- l'objet informationnel à trier sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. La liste des constituants de l'objet informationnel à trier apparaît à l'écran, premièrement dans son ordre initial et ensuite dans le nouvel

ordre de classement que lui confère l'opération.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

N) L'opération de **classement**

- l'objet informationnel à classer sort de son lieu de rangement ou de classement.
- il se dirige vers le lieu de rangement de l'objet informationnel dans lequel il doit être classé.
- l'objet dans lequel doit être classé l'objet informationnel sort de son lieu de rangement, on y dépose l'objet à classer.
- l'objet dans lequel on vient de classer retourne dans son lieu de rangement.

O) L'opération de **rangement**

- l'objet informationnel à ranger sort de son lieu de rangement ou de classement.
- il se dirige vers le lieu de rangement dans lequel il doit être rangé.

P) L'opération de **replacement**

- les uns après les autres, les objets informationnels à replacer se dirigent vers leur dernier lieu de rangement ou de classement.

Q) L'opération de **décision**

- l'objet informationnel à partir duquel la décision doit être prise, sort de son lieu de rangement ou de classement et se dirige vers la table de travail.
- un zoom est effectué afin de simuler l'opération proprement dite. Un squelette dépendant du type d'objet informationnel apparaît à l'écran de même que ses attributs.
- l'utilisateur entre sa réponse concernant la décision à prendre.
- le zoom se termine et l'objet informationnel est déposé sur la table de travail.

6.4 Conséquences sur le schéma conceptuel du bureau

Il faut modifier le schéma conceptuel du bureau de telle manière que le système ait à sa disposition toute l'information nécessaire pour le gestion de l'animation des icônes à l'écran. Cette information lui permettra de connaître pour chaque objet, l'emplacement de son icône à l'écran et les dimensions de celle-ci.

Nous convenons que les objets informationnels n'apparaissent pas à l'écran entre deux opérations primitives et ce, pour des raisons de clareté. Il ne sera donc pas nécessaire d'ajouter des attributs à l'entité INFORMATION.

Pour les entités OBJET_RANGEMENT, OBJET_INTERFACE, OUTIL_TRAVAIL, l'emplacement et les dimensions d'une icône y sont mémorisés sous la forme de quatres attributs : X1, Y1, X2, Y2 (cfr. figure 6.5).

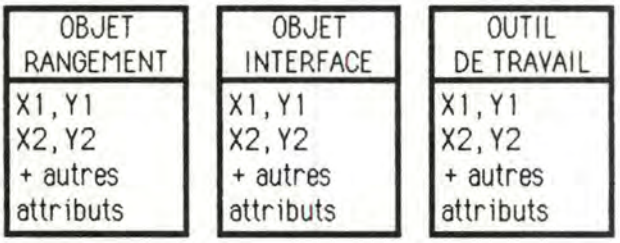


Fig 6.5 : Attributs graphiques ajoutés aux entités

Le couple (X1,Y1) spécifie le coin supérieur gauche de l'icône à l'écran :

- X1 représente l'abscisse du point (X1,Y1).
- Y1 représente l'ordonnée du point (X1,Y1).

Le couple (X2,Y2) spécifie le coin inférieur droit de l'icône à l'écran :

- X2 représente l'abscisse du point (X2,Y2).
- Y2 représente l'ordonnée du point (X2,Y2).

Nous imposons comme contrainte que les couples (X1,Y1) et (X2,Y2) spécifient un point de l'écran.

Chapitre 7
Implémentation du prototype

CHAPITRE 7 : IMPLEMENTATION DU PROTOTYPE

7.1 Introduction

Ce chapitre est destiné à la description de notre implémentation de l'animation graphique des tâches de bureau. Nous précisons de suite que cette description sera la plus intuitive possible, de manière à donner au lecteur une vision globale de l'implémentation que nous proposons et cela sans rentrer dans des détails techniques inutiles pour le moment. Le développement complet du module SIMULATION est donné en annexe. Les autres modules qui seront présentés ci-dessous ne seront pas implémentés.

7.2 La découpe logique de l'application

Lorsque l'analyste désire réaliser la simulation graphique d'une tâche de bureau, il doit veiller à deux choses. Premièrement, pour un bon déroulement de la tâche, il faut que tous les objets du bureau soient définis dans le système. Deuxièmement, il faut que la tâche qu'il veut exécuter soit également définie correctement dans le système.

Ainsi, avant de commencer l'application, il pourra éventuellement mettre à jour l'environnement du bureau en ajoutant des objets dont sa tâche aura besoin ou en supprimant d'autres objets et il pourra également introduire le texte de sa tâche. L'utilisation d'un langage non procédural, rappelons-le, lui permettra d'introduire ses différentes phrases dans un ordre quelconque.

De façon à éviter une exécution d'une tâche erronée, dès le départ le système effectuera le maximum de contrôles sur le texte introduit par l'analyste. Les erreurs éventuelles lui étant signalées. Les contrôles effectués sont ceux décrits au point 5.4.

Lorsque le système ne détectera plus d'erreurs, la simulation graphique pourra alors débuter.

L'architecture logique reprenant ces différentes notions est donnée à la figure 7.1.

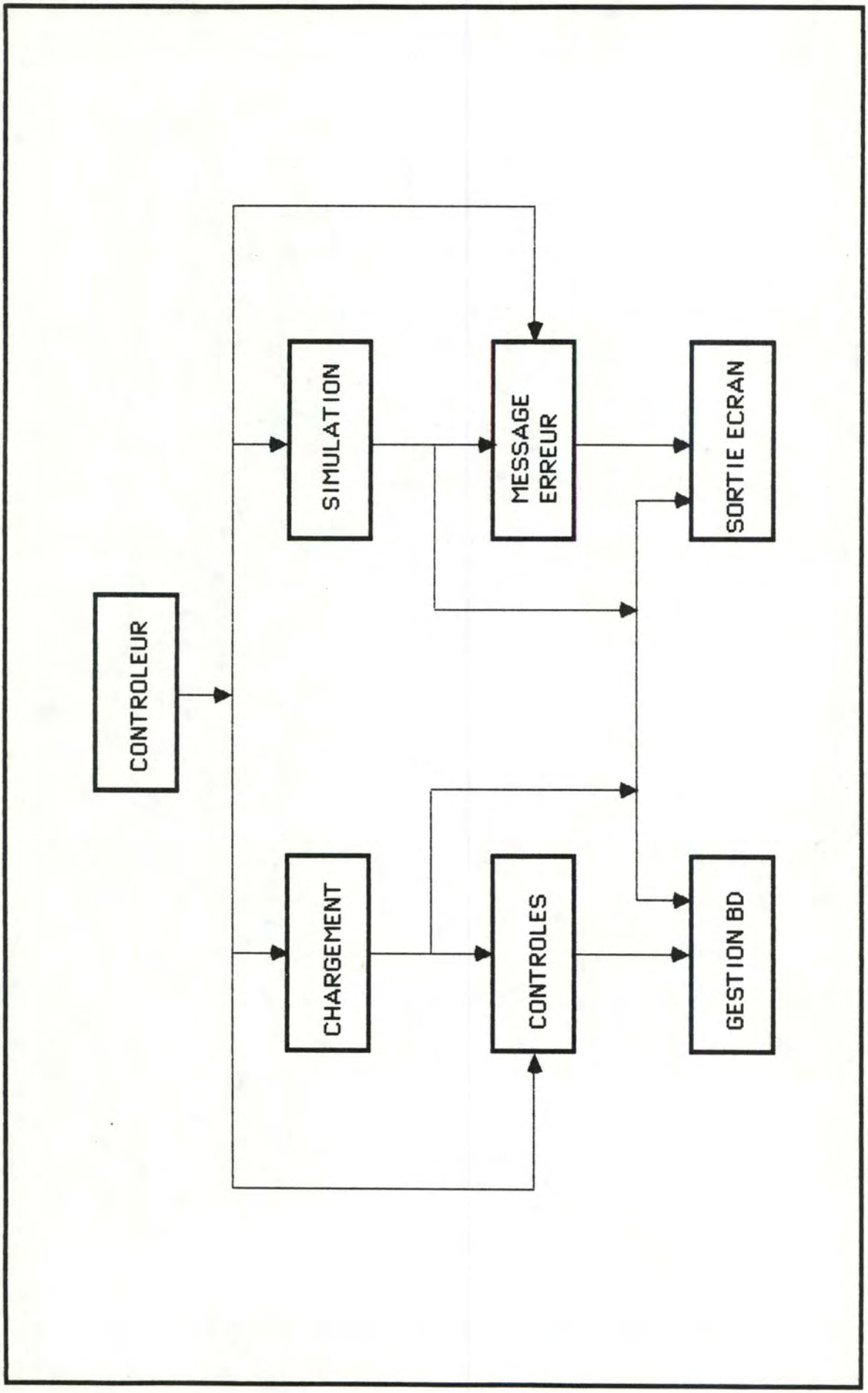


Fig. 7.1 : Architecture générale de l'application

7.2.1 Le module CONTROLEUR

Le module CONTROLEUR prend en charge l'exécution complète de la simulation. Il doit demander à l'analyste ce qu'il veut faire :

- encoder une tâche,
- mettre à jour l'environnement du bureau,
- exécuter une tâche.

Si l'analyste veut exécuter une tâche, le CONTROLEUR doit veiller à ce qu'elle ait été contrôlée au préalable.

7.2.2 Le module CHARGEMENT

Le module chargement doit saisir les informations introduites par l'analyste. Ces informations peuvent être de deux types :

- soit, l'analyste met à jour l'environnement du bureau et le module CHARGEMENT doit les mémoriser dans la base de données du bureau.
- soit, l'analyste encode une tâche et le module CHARGEMENT doit mémoriser les informations introduites dans la base de données du langage. Si l'analyste le désire, il peut demander au système de contrôler les informations au fur et à mesure qu'il les introduit.

7.2.3 Le module CONTROLES

Le module CONTROLES doit pouvoir contrôler deux choses :

- soit un sous-ensemble de phrases introduites par l'analyste concernant une tâche spécifiée,
- soit l'ensemble des informations concernant une tâche spécifiée.

Si des erreurs sont détectées, le module CONTROLES le signalera au module qui l'utilise.

7.2.4 Le module GESTION BD

Le module GESTION BD est chargé de prendre en compte toutes les demandes d'accès au système d'information qui peuvent lui parvenir. Ce module est le seul qui connaisse les détails d'implémentation du S.G.D. (Système de Gestion de Données) utilisé.

7.2.5 Le module MESSAGE ERREUR

Le module MESSAGE ERREUR doit afficher à l'écran le message d'erreur correspondant au code qui lui est transmis.

7.2.6 Le module SORTIE ECRAN

Le module SORTIE ECRAN est le seul qui a des interactions avec le terminal de l'analyste. C'est lui notamment qui connait le logiciel qui gère le graphisme.

7.2.7 Le module SIMULATION

Le module SIMULATION reçoit le contrôle du module CONTROLEUR lorsque l'analyste veut simuler une tâche et cette tâche a été jugée correcte par le module CONTROLES. Le module SIMULATION est le seul que nous implémenterons car il se trouve être le plus pertinent pour notre travail. C'est lui qui permettra d'évaluer le caractère faisable ou non du projet tel qu'il a été défini au chapitre1.

Etant le plus important, nous le décrivons plus complètement tout en évitant les détails. Les spécifications complètes du module SIMULATION se trouvent en annexe (annexe 2 et annexe 3).

Le travail de ce module peut se résumer en quatre fonctions :

- il doit rechercher dans la base de données du langage l'opération suivante qui doit être simulée,
- il doit procéder sur cette opération primitive, les contrôles qui n'ont pu être réalisés par le module CONTROLES,
- il doit mettre à jour la base de données du bureau d'après le type et les informations de l'opération simulée,
- il doit simuler graphiquement l'opération primitive à l'écran.

Ces quatres fonctions sont réalisées respectivement par les modules SEQUENCEUR, CONTROLES_A_PRIORI, MANIPULATION_BD et GRAPHIQUE.

L'architecture logique du module SIMULATION est donnée à la figure 7.2.

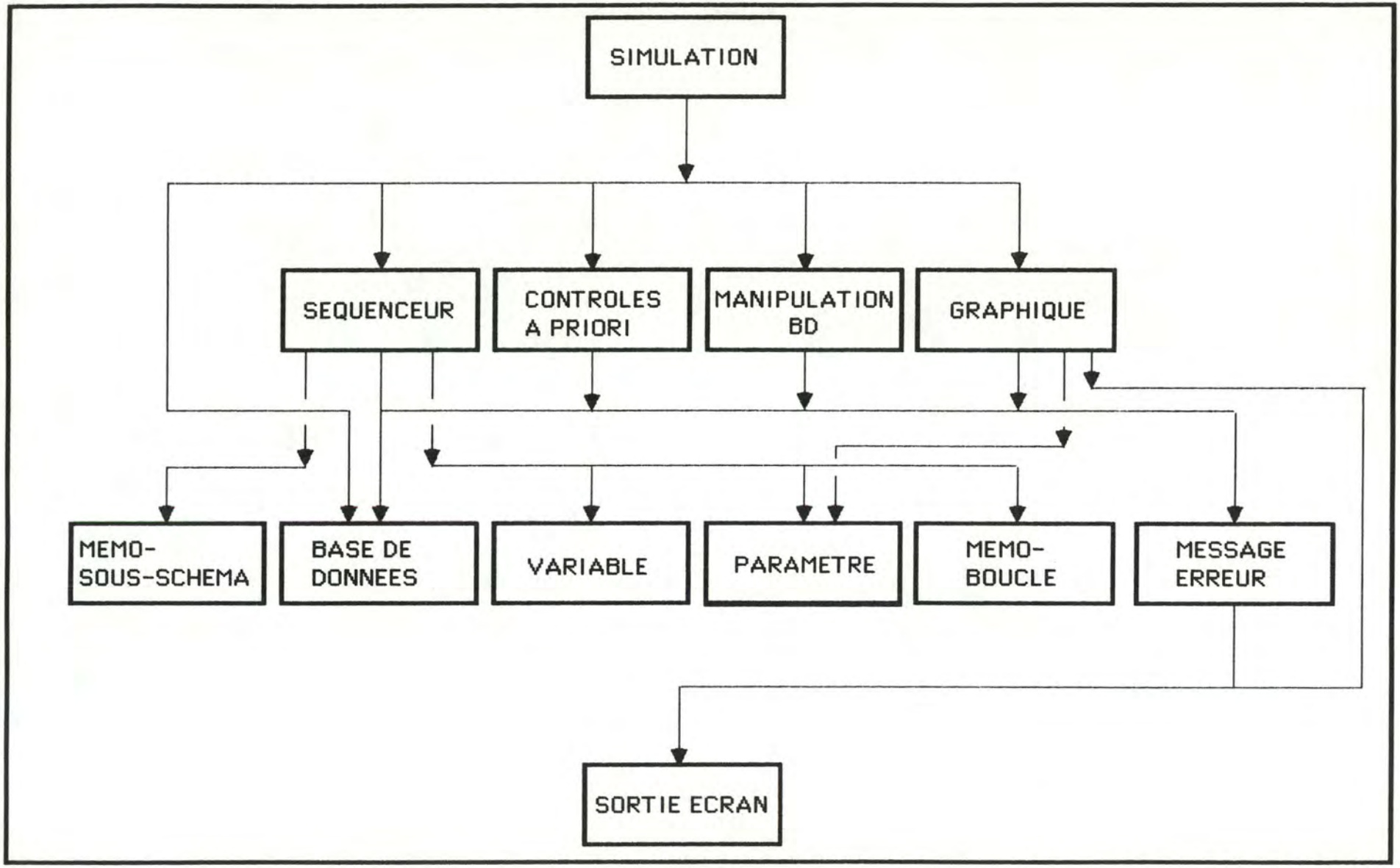


Fig. 7.2 : Architecture logique du module SIMULATION

Le fonctionnement du module SIMULATION repose sur le module SEQUENCEUR qui est chargé de retrouver l'opération primitive qui doit être simulée à un instant donné. L'originalité de ce module est l'abstraction qu'il fait sur les composants boucle, condition, sous-schéma et tâche. De cette façon, le module SIMULATION doit juste lui demandé : *Donnez moi l'opération primitive suivante à simuler*. A quoi, le module SEQUENCEUR répond soit par:

- l'opération primitive suivante à simuler,
- le message : *C'est la fin de la tâche car il n'y a plus d'opération primitive à simuler.*

L'architecture logique du module SEQUENCEUR est donné à la figure 7.3 ; il n'est pas utile de décrire ici chacun des modules qui y interviennent car ils sont spécifiés en annexe (annexe 3).

Toutes les structures de données en C, de même que toutes les constantes utilisées dans les programmes sont décrits à l'annexe 2. La transformation des schémas entité-association, exposés dans ce mémoire, en schémas conformes au S.G.D. (Système de Gestion de Données) est décrite à l'annexe 4. La représentation des structures est donnée à l'annexe 5. Enfin, pour terminer les annexes, les programmes résultant de l'implémentation du séquenceur sont donnés à l'annexe 6 : ils y sont classés par modules.

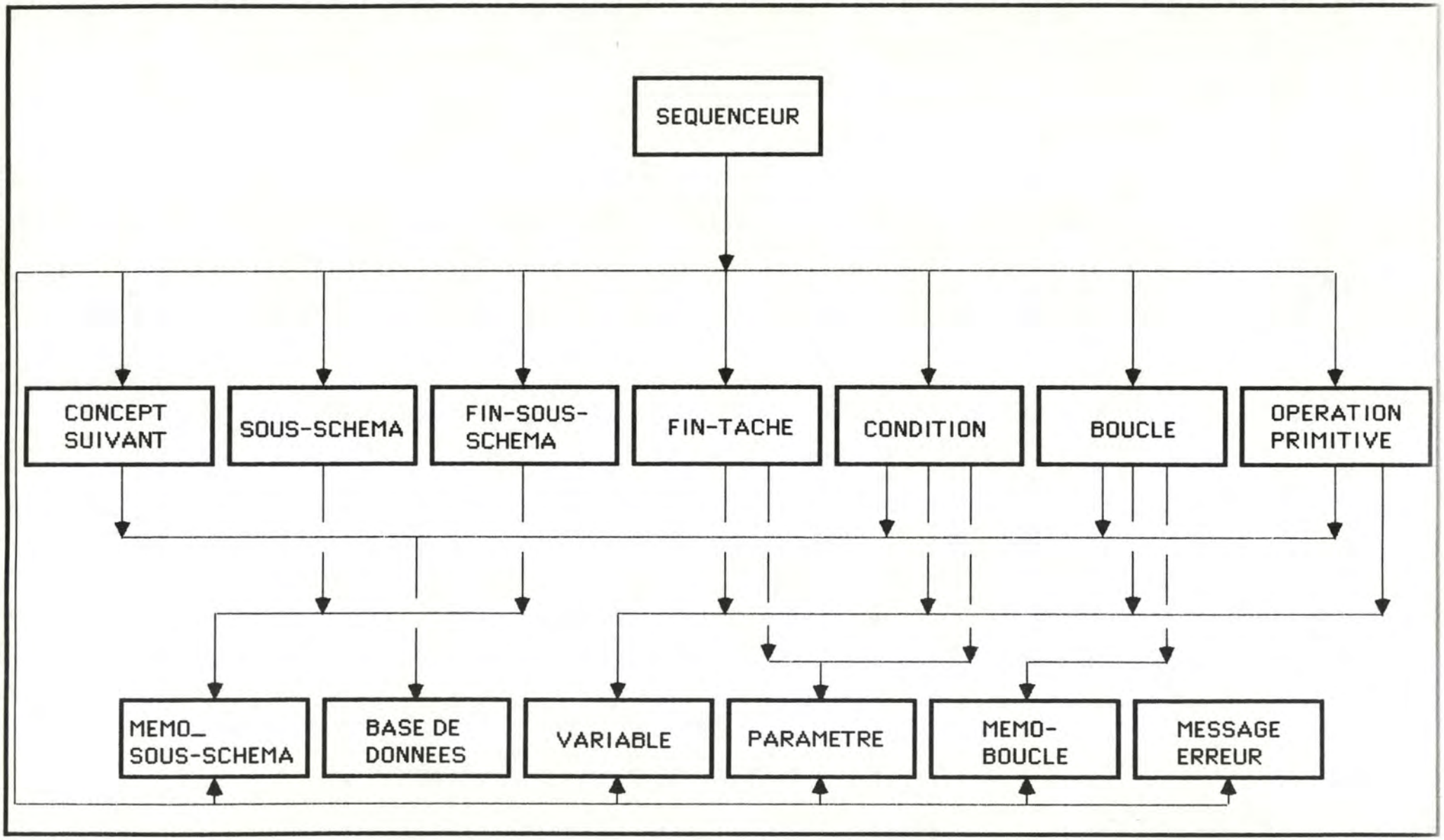


Fig. 7.3 : Architecture logique du module séquenceur

Chapitre 8
Conclusion

CHAPITRE 8 : CONCLUSION

Dans cette conclusion, nous nous proposons d'établir une synthèse du travail réalisé dans ce mémoire. Nous évaluerons égalemment le graphisme et le modèle de représentation de données que nous avons utilisés. Nous terminerons cette conclusion en donnant les perspectives possibles pour la suite du projet ainsi qu'une évaluation de notre langage de description de tâche de bureau.

8.1 Synthèse

Le problème de la communication entre une personne travaillant dans un bureau et un informaticien chargé d'automatiser totalement ou partiellement le travail de cette personne constitue la base du projet dans lequel s'insère notre travail.

Ce projet propose à l'analyste un outil de spécification de tâches de bureau ainsi qu'un simulateur graphique qui serviront à montrer à l'utilisateur :

- d'une part la vision que l'analyste a du travail de l'utilisateur,
- d'autre part, l'impact qu'aura l'automatisation proposée de ce travail.

Cette mise au point des spécifications ne se faisant pas du premier coup, des retours en arrière seront fréquents : l'utilisateur détecte des erreurs ou des inconforts dans ce qui lui est montré et l'analyste doit les corriger. Cette méthode de travail peut être améliorée en recourant au prototypage : l'analyste construit un prototype des tâches de l'utilisateur et ce dernier évalue la qualité de ce prototype grâce à l'animation graphique proposée.

Pour construire ce prototype, nous avons dû successivement analyser les objets présents dans le bureau (chapitre 3) et définir un langage non procédural de haut niveau (chapitre 4). Ce langage fournit à l'analyste un moyen de décrire les objets et les traitements pouvant être faits sur ceux-ci.

Si l'originalité et le point important de ce travail est l'**animation graphique**, elle ne constitue cependant qu'une interface, les cinq premiers chapitres constituant l'analyse et la modélisation de ce que le graphisme va montrer. C'est donc seulement dans le chapitre 6 que nous présentons le graphisme. Nous y introduisons la technique d'animation choisie ainsi que la description graphique de chacune des opérations primitives.

Nous terminons ce mémoire par un chapitre consacré à l'implémentation que nous avons adoptée.

8.2 Evaluation

8.2.1 Le modèle Entité-Association et les langages orientés objets

L'utilisation du modèle entité-association a été de pure convenance. Il serait peut-être plus intéressant de travailler complètement avec les langages orientés objets car l'analyse du problème s'y prête bien. De plus, ces langages offrent généralement des mécanismes d'ajout de nouveaux objets et d'opérations primitives s'appliquant à ceux-ci, ce qui sera utile pour l'extension de notre travail (cfr. infra 8.3.3).

8.2.2 Evaluation de notre langage

Certaines limites de notre langage ne nous sont apparues qu'au moment de l'implémentation. Il nous semble indispensable de les citer :

- il n'est pas possible de dupliquer facilement un objet informationnel structurant (dossier, fichier, pile) si ce n'est au moyen d'artifices peu élégants (cfr. annexe 1, exemple 3).
- il n'est pas possible de ranger quelque chose dans la table de travail, ce qui semble assez pauvre pour respecter la réalité.

Ce sont deux limites de notre langage de description de tâche de bureau. Toutefois, ceci semble normal dans la mesure où nous n'avons jamais prétendu être exhaustifs puisque nous nous situons dans une approche par prototypage.

La faisabilité d'implémentation d'un outil utilisant un tel langage a cependant été montrée au chapitre 7.

8.2.3 Evaluation du graphisme utilisé

Le graphisme constitue le point le plus important du projet. La modélisation du bureau peut être aussi complète que possible, si l'interface graphique n'est pas performante, le projet est irréalisable. L'outil graphique doit être suffisamment précis pour montrer un maximum de détails à l'utilisateur. En effet, sur l'écran tous les objets de bureau doivent pouvoir être dessinés et identifiés par l'utilisateur.

Il faut non seulement représenter les objets du bureau mais leur disposition à l'écran est certainement tout aussi importante. L'utilisateur doit se trouver confronté à quelque chose de familier et qu'il doit pouvoir identifier avec précision. Il faudrait donc prévoir la possibilité pour l'analyste de définir l'emplacement des différents objets à l'écran. Dans cet ordre d'idée, il ne s'agit pas non plus de représenter les armoires du bureau symboliquement par une seule armoire dessinée à l'écran. Il faut également représenter tous les tiroirs des armoires et l'utilisateur doit pouvoir les identifier.

La réussite de ce projet dépend en majeure partie de la précision avec laquelle la simulation graphique est faite. En effet, ce projet est destiné à réduire le problème de la communication; il ne s'agit donc pas de noyer l'utilisateur dans une représentation de son bureau qu'il ne reconnait pas.

L'implémentation, réalisée avec le logiciel MS-WINDOWS, semble indiquer d'une part, que cet outil n'a pas été conçu pour de l'animation graphique pure mais plutôt pour la gestion de fenêtres et que d'autre part, que le type d'animation choisi, le déplacement d'icônes, n'est pas ce qu'il y a de plus parlant pour l'utilisateur.

8.3 Perspectives

Le travail réalisé dans ce mémoire ne constitue qu'un prototype d'une animation graphique de tâches de bureau. Il a pour objectif de réaliser une première étude dans un domaine encore fort peu exploré actuellement. Cette première approche aura permis de découvrir certains domaines qu'il serait intéressant d'approfondir. Nous citons les principaux ci-dessous.

8.3.1 Communication entre tâches

Dans notre modèle, l'envoi ou la réception d'informations se fait par les boîtes IN et OUT. Cependant, dans notre cas, cette information reste associée à ces boîtes. Dans la réalité, l'information est destinée à être envoyée ou reçue respectivement vers l'extérieur ou de l'extérieur du bureau Il serait donc intéressant de modéliser la communication d'informations entre tâches.

Rappelons que nous n'avions envisagé que la participation d'une seule personne à la réalisation d'une tâche. Dans le cas où une ou plusieurs personnes travaillent à une même tâche de bureau, nous proposons de modéliser ce travail par une communication entre tâche. Chacune des personnes contribuant à réaliser la tâche commune en exécutant sa propre tâche, c'est-à-dire le travail qui lui est imparti. Une tâche est donc liée au concept de personne plutôt qu'à celui de bureau. Elle représente le travail réalisé par une personne dans un bureau.

Ces deux aspects montrent que la communication entre tâches est importante à réaliser. A partir de notre modélisation, la solution suivante pourrait être proposée. Celle-ci n'a pas été approfondie et ne constitue qu'une piste de réflexion.

a) Communication écrite

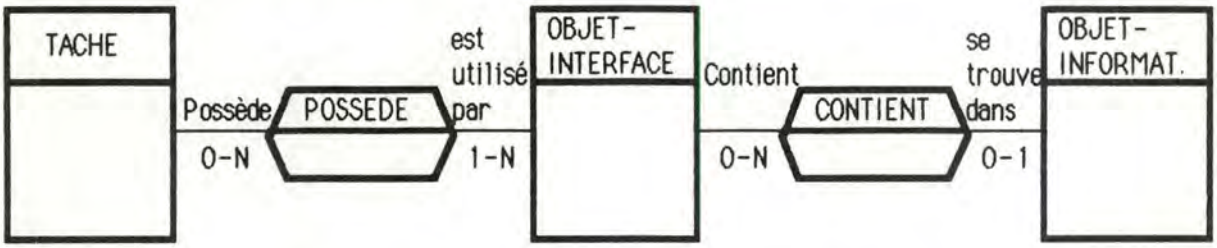


Fig. 8.1 : Modélisation de la communication écrite

La figure 8.1 montre comment la communication d'informations entre tâches pourrait être résolue. Chaque tâche décrite dans le système est associée à ses objets d'interface. Dans le cas d'une communication écrite, ces objets seraient la boîte IN et la boîte OUT.

Une tâche désirant communiquer un objet informationnel à une autre tâche va simplement le déposer dans la boîte IN de cette dernière tâche. Cette communication pouvant se faire directement ou via la boîte OUT de la tâche émettrice.

Cette solution pourrait être gérée par les opérations primitives suivantes:

- retirer un objet informationnel de la boîte IN,
- déposer un objet informationnel dans la boîte IN d'une autre tâche,
- déposer un objet informationnel dans la boîte OUT de la tâche,
- transmettre le contenu de la boîte OUT vers les boîtes IN des tâches destinatrices.

b) La communication orale

Bien que rarement utilisée de manière systématique, nous apportons maintenant un élément de solution concernant la communication orale.

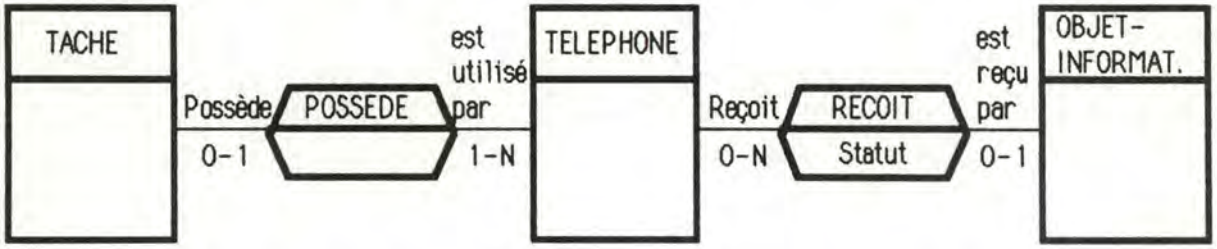


Fig. 8.2 : Modélisation de la communication orale

Le traitement des échanges téléphoniques entre tâches pourrait se résoudre de la même manière que pour la communication écrite. La figure 8.2 montre que chaque tâche est associée à son téléphone. Lorsqu'une tâche envoie un message téléphonique à une autre tâche, ce message est associé au téléphone de cette

dernière tâche avec le statut EN ATTENTE. Cela signifie qu'un message est arrivé mais qu'il n'est pas encore traité. Dans la réalité, cela équivaut au téléphone qui sonne.

La tâche réceptrice d'un tel message peut le traiter et elle peut soit :

- le détruire s'il n'y a pas de trace écrite conservée dans le bureau,
- le conserver associé au téléphone avec le statut RECU. Ce cas modélise une conversation téléphonique avec trace écrite.

Cette solution pourrait être gérée par les opérations primitives suivantes :

- recevoir un message téléphonique avec ou sans trace écrite,
- envoyer un message téléphonique.

8.3.2 L'enchaînement de tâches

La communication entre tâches induit que plusieurs tâches doivent être décrites dans le système et qu'il y a un certain enchaînement entre ces tâches. La modélisation de cet enchaînement devrait permettre une simulation d'une exécution parallèle des tâches : il y aurait une exécution concurrente entre les tâches. Ainsi, chaque tâche pourrait être soit :

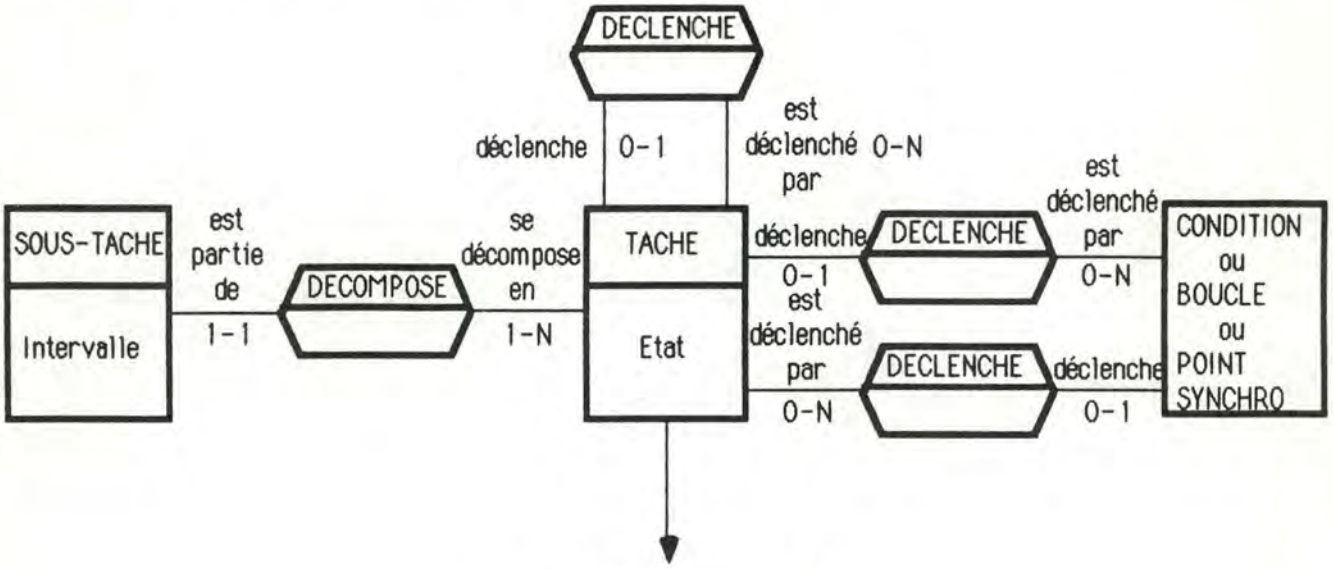
- dans l'état actif : la tâche s'exécute (une seule tâche est active à un instant donné),
- dans l'état inactif : la tâche n'a pas commencé son exécution ou l'a terminée,
- dans l'état interrompu : la tâche a passé la main à une autre tâche.

Pour chaque tâche décrite dans cet enchaînement, il faudrait en plus donner :

- la précondition de déclenchement de la tâche : exprime la ou les conditions qui doivent être remplies pour que la tâche puisse commencer son exécution. Exemple : attendre que tel message soit reçu ou que telle tâche ait terminé son exécution.

- la postcondition d'arrêt : exprime la condition d'arrêt de la tâche. Exemple : à la fin de cette tâche, le dossier de l'étudiant est dans l'état VERIFIE. Lorsque cette condition est remplie, la tâche est soit terminée et passe de l'état actif à celui d'inactif soit elle est interrompue et elle passe alors de l'état actif à celui d'interrompue. Cette condition pourrait être exprimée sous forme d'intervalle spécifiant après quelle partie de la tâche, celle-ci peut être interrompue.

Dans la modélisation de notre langage (cfr. figure 5.1), cet enchaînement peut facilement y être ajouté comme illustré à la figure 8.3.



SCHEMA DE LA FIGURE 5.1

Fig. 8.3 : Modélisation de l'enchaînement de tâches

8.3.3 La généralisation

Le modèle de bureau développé dans ce mémoire est inévitablement incomplet car il est difficile sinon impossible d'envisager tous les objets de bureau, tous les objets informationnels ainsi que toutes les opérations primitives pouvant être réalisées sur ces objets. Dès lors un mécanisme de généralisation des objets et des opérations serait intéressant. Par exemple, si l'analyste veut travailler avec un objet x qui n'est pas décrit dans notre modèle, il doit pouvoir le créer ainsi que le type auquel cet objet appartient.

Ce problème est très complexe à résoudre car il faut travailler à un niveau supérieur avec un **meta-langage**. Grâce à ce meta-langage, le système devrait pouvoir accepter une nouvelle opération primitive travaillant sur un nouvel objet informationnel.

En conclusion, le modèle idéal serait celui qui permettrait à l'analyste :

- de décire ses opérations primitives,
- de décrire ses propres objets,
- de décrire un enchaînement entrelacé de tâches,
- de faire communiquer les tâches.

BIBLIOGRAPHIE

[ALAV 84] : ALAVI, M. (1984), "An assessment of the prototyping approach to information systems development", *Communications of the ACM*, vol. 27, n° 6, pp. 556-563.

[BLAS 82] : de BLASIS, J.P. (1982), *La bureautique : outils et applications*, Les éditions d'organisation, Paris.

[BOPI 83] : BODART, F. and PIGNEUR, Y. (1983), *Conception assistée des applications informatiques : étude d'opportunité et analyse conceptuelle*, Masson, Presses universitaires de Namur.

[BUDD 84] : BUDDE, R. , KUHLENKAMP, K. , MATHIASSEN, L. and ZULLIGHOVEN, H. (1984), *Approaches to prototyping*, Springer-Verlag, New-York.

[CANN 81] : CANNING, R.G. (1981), "Developping systems by prototyping", *EDP Analyser*, vol. 19, n° 9, pp. 1-12.

[CARB 81] : CARBONELL, J.G. (1981), "Default reasoning and inheritance mechanisms on type hierarchies", *ACM SIGPLAN notices*, vol.16, N° 1, pp. 107-109.

[CHEN 76] : CHEN, P.P.S. (1976), "The entity-relationship model - toward a unified view of data", *ACM Transactions On Database Systems*, vol.1, n° 1, pp. 9-36.

[CLAN 83] : CLANTON, C. (1983), "The future of metaphor in man-computer systems", *Byte*, n° 12, pp. 263-279.

[DACH 86] : DACHOUFFE, N. (1986), *Spécification et implémentation d'un modèle d'information de bureau*, Mémoire inédit, Institut d'informatique, F.N.D.P., Namur.

[ELMA 85] : ELMASRI, R. , WEEL, D. , REYER, J. and HEUNER, A. (1985), "The category concept : an extension to the entity-relationship model", *Data & Knowledge Engineering*, n° 1, pp. 75-116.

[HAIN 86] : HAINAUT, J.L. (1986), *Conception assistée des applications informatiques : conception de la base de données*, Masson, Presses Universitaires de Namur.

[HINE 85] : HINES, V.D. (1985), *Office automation : tools and methods for system building*, John Wiley & Sons.

[HIRS 85] : HIRSCHHEIM, R.A. (1985), *Office Automation : a social and organizationnal perspective*, John Wiley & Sons.

[HISA 84] : HIGGINGS, C. A. and SAFAYENI, F.R. (1984), "A critical appraisal of task taxonomies as a tool for studying office activities", *ACM Transactions on Office Information Systems* , vol. 2, n° 4, pp. 331-339.

[KEEN 78] : KEEN, P.G.W. and SCOTT MORTON, M. (1978), *Decision Support System : an organizationnal perspective*, Addison-Wesley.

[KEEN 81] : KEEN, P.G.W. (1981), "Information Systems and Organizationnal Change", *Communications of the ACM*, vol. 24, n° 1, pp. 24-33.

[LAPA 86] : LANGELEZ, F. and PARIS, C. (1986), *Etude et évaluation d'un outil de prototypage*, Mémoire inédit, Institut d'informatique, F.N.D.P., Namur.

[LEAV 65] : LEAVITT, H.J. (1965), *Applying organizational change in industry. structural, technological and humanistic approaches. Handbook of organizations*, Rand Mc Nally, Chicago.

[LINE 86] : LIPECK, W. and NEUMANN, K. (1986), "Modelling and manipulating objects in geoscientific databases", *5 th International Conference on E/R Approach*, Dijon, Novembre, pp. 105 - 124.

[META 86] : METAIS, T. (1986), *Manuel de référence OMEGA v1.0* , EDF-GDF STI/DEMA.

[RADU 84] : RADUCHEL, W.J. (1984), "A professional's perspective on user-friendliness", *Byte,* n°5, pp. 101-106.

[RENT 82] : RENTSCH, T. (1982), "Object oriented programming", *ACM SIGPLAN notices*, vol. 17, n°9, pp. 51-57.

[ROBE 86] : ROBERT, S. (1986), *Analyse et implémentation d'un environnement de rangement d'informations*, Mémoire inédit, Institut d'informatique, F.N.D.P., Namur.

[ROBS 81] : ROBSON, D. (1981), "Object oriented software systems", *Byte* , n° 8, pp. 74-86.

[ROGO 83] : GOLDBERG, A. and ROBSON, D. (1983), *Smalltalk-80 : the language and its implementation*, Addison-Wesley.

[SHNE 77] : SHNEIDERMAN, B., MAYER, R., McKAY, D. and HELLER, P. (1977), "Experimental investigations of the utility of detailled flowcharts in programming", *Communications of the ACM*, vol. 20, n° 6, pp. 373-381.

[SMSM 77] : SMITH, J.M. and SMITH, D.C.P. (1977), "Database abstractions : aggregation and generalization", *ACM Transactions On Database Systems*, vol. 2, n° 2, pp. 105-133.

[XERO 81] : The Xerox Learning Research Group (1981), "The smalltalk-80 system", *Byte* , n°8, pp. 36-48.

[WARF 83] : WARFIELD, R.W. (1983), "The new interface technology", *Byte*, n°12, pp. 218-230.

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Animation graphique de tâches de bureau

- Annexes (1ère partie) -

Mémoire présenté par
Marc Simoens et Olivier Van Pevenaeyge
en vue de l'obtention du titre de
Licencié et Maître en Informatique.

Promoteur : Mr. Roland Lesuisse.

Année académique 1986 - 1987.

TABLE DES MATIERES

Annexe 1
Description de tâches de bureau

ANNEXE 1 : DESCRIPTION DE TACHES DE BUREAU

Nous allons décrire dans cette annexe l'application que nous avons fait de notre langage à quelques tâches de bureau typiques. Le langage a été défini de manière précise au chapitre 4.

1.1 Description des tâches

Le cadre général des tâches de bureau que nous allons envisager est le traitement des demandes d'inscription de la part des étudiants auprès du secrétariat administratif de l'Institut d'Informatique.

Les descriptions qui vont suivre, bien que modifées afin de montrer les possibilités et les limites du langage, sont largement inspirées de N. Dachouffe [DACH 86, Annexe 1].

1.1.1 Tâche 1 : Prise de contact de la part de l'étudiant

La secrétaire reçoit un contact téléphonique de la part de l'étudiant. Elle prépare une réponse à envoyer, consistant en une lettre d'acceptation et une demande d'inscription en double exemplaire. Elle envoit la réponse à l'étudiant.

1.1.2 Tâche 2 : Vérification et classement des demandes d'inscription

La secrétaire reçoit un dossier de la part de l'étudiant comprenant une lettre de présentation et une demande d'inscription en double exemplaire. Après avoir consulté la lettre de présentation, elle vérifie si la demande d'inscription est complète ou non.

Si la demande n'est pas complète, elle renvoit le dossier à l'étudiant après y avoir joint une lettre de demande de renseignements.

Si la demande est complète, elle décide si l'étudiant doit ou non présenter un ou plusieurs examens d'entrée. En fonction de cette décision, elle communique à l'étudiant les documents adéquats.

La secrétaire termine par classer un exemplaire de la demande d'inscription dans un fichier des demandes d'incriptions et le dossier restant dans le fichier des dossiers des étudiants.

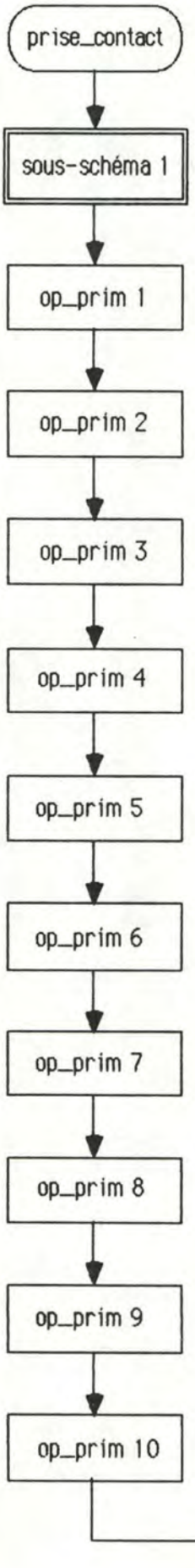
1.1.3 Tâche 3 : Rentrée académique

A la rentrée académique, la secrétaire reprend le fichier des demandes d'inscription, en fait une copie et transmet cette copie au secrétariat central.

1.2 Description graphique des tâches

Nous allons dans ce point décrire les tâches grâce aux outils graphiques suggérés au point 4.5.5.

1.2.1 Tâche 1 : Prise de contact de la part de l'étudiant



Description de la tâche de traitement d'une prise de contact.

La secrétaire reçoit oralement un contact dont elle garde une trace écrite.

La secrétaire crée un dossier vide pour répondre à l'étudiant.

La secrétaire rédige une lettre d'acceptation.

La secrétaire reproduit en 2 exemplaires un formulaire de demande d'inscription vierge.

La secrétaire classe la lettre d'acceptation dans le dossier de réponse.

La secrétaire classe le premier formulaire dans le dossier de réponse.

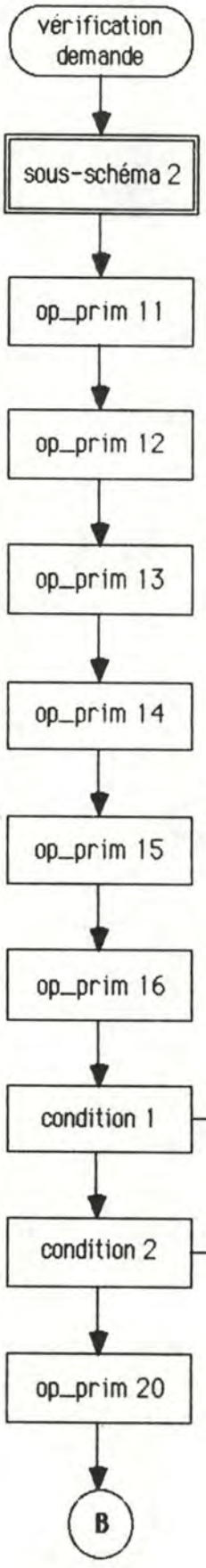
La secrétaire classe le deuxième formulaire dans le dossier de réponse.

La secrétaire envoit le dossier de réponse à l'étudiant.

La secrétaire classe la trace écrite du contact dans le fichier des contacts.

La secrétaire range le formulaire de demande d'inscription vierge.

I.2.2 Tâche 2 : Vérification et classement des demandes d'inscription



Description de la tâche de vérification d'une demande d'inscription.

La secrétaire reçoit par le courrier extérieur le dossier de l'étudiant.

La secrétaire feuillète le dossier.

La secrétaire vérifie la complétude du premier formulaire.

La secrétaire vérifie la complétude du deuxième formulaire.

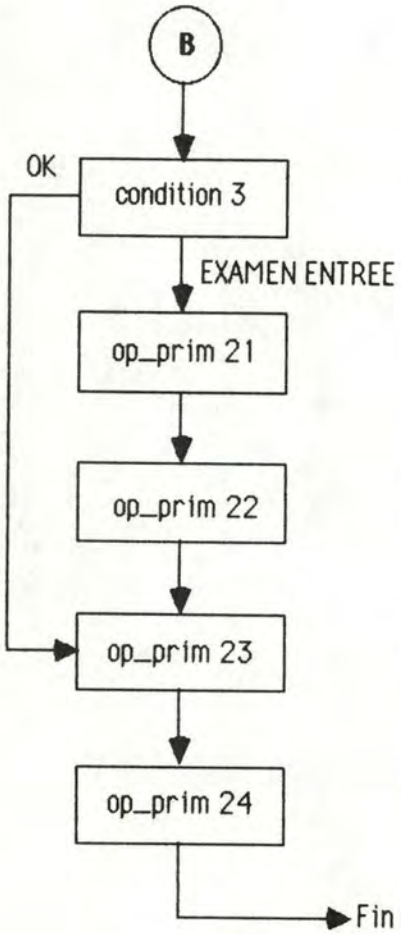
La secrétaire replace le premier formulaire dans le dossier.

La secrétaire replace le deuxième formulaire dans le dossier.

On teste le résultat de la première vérification.

On teste le résultat de la deuxième vérification.

La secrétaire décide si l'étudiant doit ou non passer des examens d'entrée.



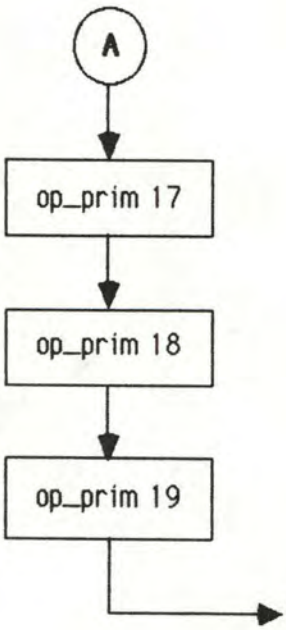
On teste le résultat de la décision.

La secrétaire rédige le document reprenant les examens d'entrée à présenter.

La secrétaire envoit le document à l'étudiant.

La secrétaire classe le premier formulaire dans le fichier des demandes d'inscription.

La secrétaire classe le dossier dans le fichier d'étudiants.

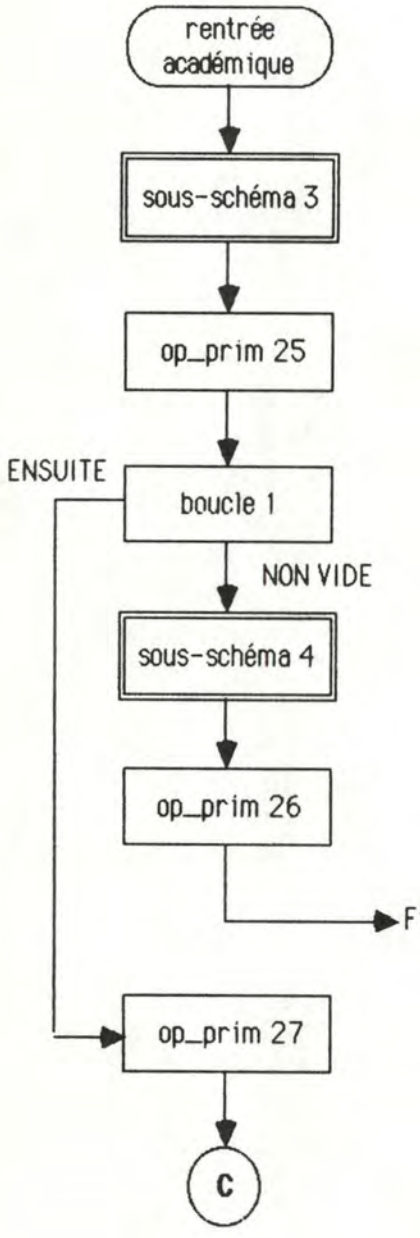


La secrétaire rédige une demande de renseignements complémentaires.

La secrétaire classe la demande de renseignement dans le dossier.

La secrétaire renvoit le dossier à l'étudiant.

1.2.3 Tâche 3 : Rentrée académique



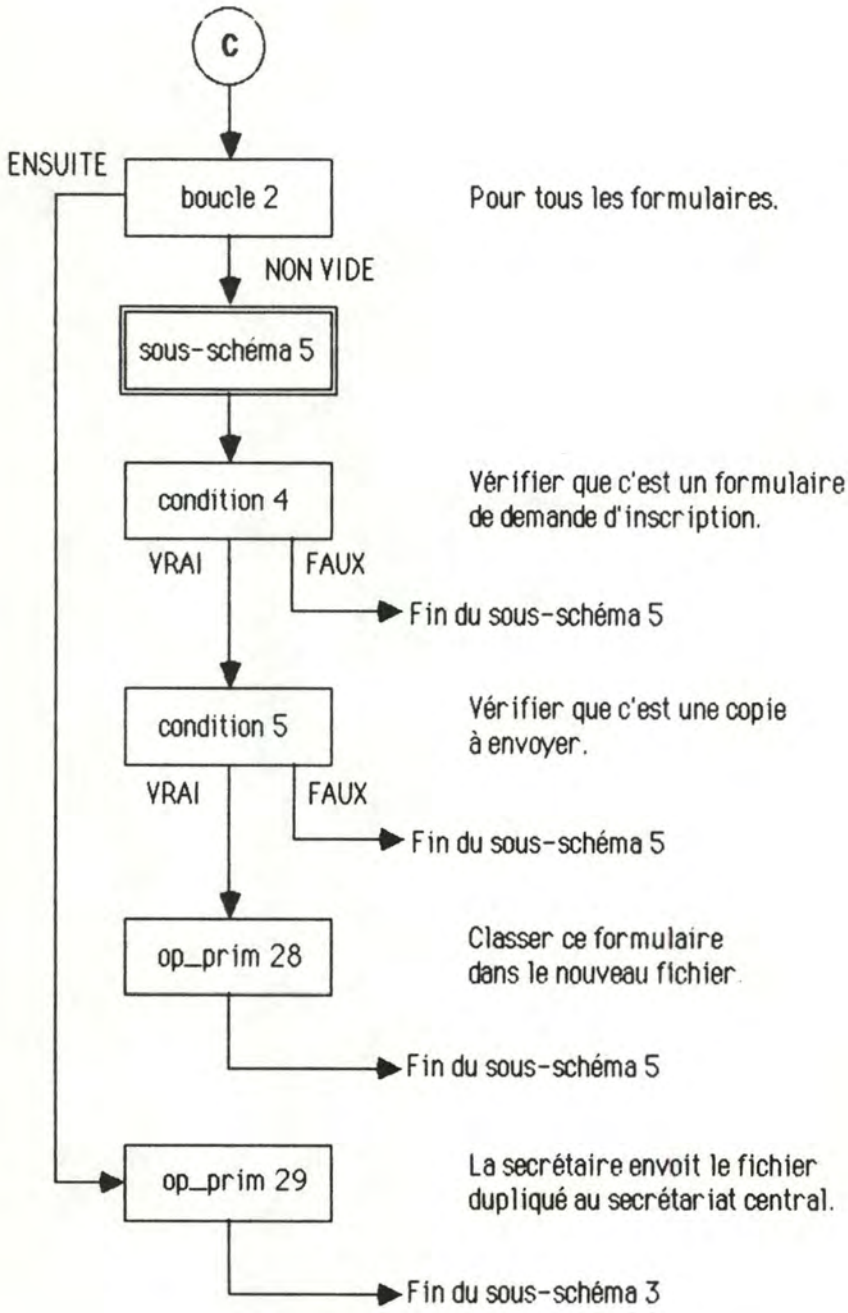
Description de la tâche de rentrée académique.

La secrétaire prépare un nouveau fichier d'inscriptions vide.

Pour chaque formulaire du fichier d'inscriptions.

Dupliquer ce formulaire en un exemplaire.

La secrétaire replace tous les formulaires qui ont été dupliqués.


C
ENSUITE
boucle 2
Pour tous les formulaires.
NON VIDE
sous-schéma 5
condition 4
Vérifier que c'est un formulaire
de demande d'inscription.
VRAI
FAUX
Fin du sous-schéma 5
condition 5
Vérifier que c'est une copie
à envoyer.
VRAI
FAUX
Fin du sous-schéma 5
op_prim 28
Classer ce formulaire
dans le nouveau fichier
Fin du sous-schéma 5
op_prim 29
La secrétaire envoit le fichier
dupliqué au secrétariat central.
Fin du sous-schéma 3

1.3 Description des tâches au moyen du langage

Nous allons maintenant écrire ces trois tâches au moyen du langage que nous avons défini au chapitre 4.

1.3.1 Tâche 1 : Prise de contact de la part de l'étudiant

```
TACHE prise_contact
DESCRIPTION : traitement par la secrétaire d'une prise de contact.
DECLENCHE SOUS-SCHEMA sous-schéma 1

SOUS-SCHEMA sous-schéma 1
DESCRIPTION : traitement par la secrétaire d'une prise de contact.
DECLENCHE PAR TACHE prise_contact
DECLENCHE OPERATION PRIMITIVE op_prim 1

OPERATION PRIMITIVE op_prim 1
DESCRIPTION : la secrétaire reçoit oralement un contact dont elle garde un trace écrite.
ENONCE : RECEVOIR ORALEMENT MESSAGE  AVEC TRACE ECRITE IDENTIFIEE PAR MESSAGE contact
         AVEC ETAT = reçu  AVEC ECHEANCE = 1987/09/01
DECLENCHEE PAR SOUS-SCHEMA sous-schéma 1
DECLENCHE OPERATION PRIMITIVE op_prim 2

OPERATION PRIMITIVE op_prim 2
DESCRIPTION : la secrétaire crée un dossier vide pour la réponse.
ENONCE : CREER DOSSIER DE TYPE ty_réponse IDENTIFIE PAR réponse
         AVEC ETAT = crée  AVEC AVEC ORDRE DE CLASSEMENT = FIFO
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 1
DECLENCHE OPERATION PRIMITIVE op_prim 3

OPERATION PRIMITIVE op_prim 3
DESCRIPTION : la secrétaire rédige la lettre d'acceptation.
ENONCE : CREER DOCUMENT DE TYPE ty_lettre IDENTIFIE PAR acceptation
         AVEC ETAT = crée
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 2
DECLENCHE OPERATION PRIMITIVE op_prim 4

OPERATION PRIMITIVE op_prim 4
DESCRIPTION : la secrétaire reproduit le formulaire de demande d'inscription vierge en 2 exemplaires.
ENONCE : REPRODUIRE 2 FOIS A PARTIR DU NUMERO 1
         FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 3
DECLENCHE OPERATION PRIMITIVE op_prim 5

OPERATION PRIMITIVE op_prim 5
DESCRIPTION : la secrétaire classe la lettre d'acceptation dans le dossier de réponse.
ENONCE : CLASSER DOCUMENT DE TYPE ty_lettre IDENTIFIE PAR acceptation
         DANS DOSSIER DE TYPE ty_réponse IDENTIFIE PAR réponse
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 4
DECLENCHE OPERATION PRIMITIVE op_prim 6
```

OPERATION PRIMITIVE op_prim 6
DESCRIPTION : la secrétaire classe le premier formulaire dans le dossier de réponse.
ENONCE : CLASSER COPIE NUMERO 1 DE
FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DANS DOSSIER DE TYPE ty_réponse IDENTIFIE PAR réponse
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 5
DECLENCHE OPERATION PRIMITIVE op_prim 7

OPERATION PRIMITIVE op_prim 7
DESCRIPTION : la secrétaire classe le deuxième formulaire dans le dossier de réponse.
ENONCE : CLASSER COPIE NUMERO 2 DE
FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DANS DOSSIER DE TYPE ty_réponse IDENTIFIE PAR réponse
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 6
DECLENCHE OPERATION PRIMITIVE op_prim 8

OPERATION PRIMITIVE op_prim 8
DESCRIPTION : la secrétaire envoit le dossier réponse à l'étudiant.
ENONCE : COMMUNIQUER DOSSIER DE TYPE ty_réponse IDENTIFIE PAR réponse
AVEC ETAT = envoyé AVEC ECHEANCE = 1987/09/05
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 7
DECLENCHE OPERATION PRIMITIVE op_prim 9

OPERATION PRIMITIVE op_prim 9
DESCRIPTION : la secrétaire classe le message de contact dans le fichier des contacts.
ENONCE : CLASSER MESSAGE contact
DANS FICHIER contacts
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 8
DECLENCHE OPERATION PRIMITIVE op_prim 10

OPERATION PRIMITIVE op_prim 10
DESCRIPTION : la secrétaire range le formulaire de demande d'inscription vierge.
ENONCE : RANGER FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DANS FARDE farde 1
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 9
DECLENCHE FIN-SOUS-SCHEMA sous-schéma 1

1.3.2 Tâche 2 : Vérification et classement des demandes d'inscription

TACHE vérification demande
DESCRIPTION : vérification par la secrétaire d'une demande d'inscription.
DECLENCHE SOUS-SCHEMA sous-schéma 2

SOUS-SCHEMA sous-schéma 2
DESCRIPTION : vérification par la secrétaire d'une demande d'inscription.
DECLENCHE PAR TACHE vérification demande
DECLENCHE OPERATION PRIMITIVE op_prim 11

OPERATION PRIMITIVE op_prim 11
DESCRIPTION : la secrétaire reçoit par le courrier extérieur le dossier de l'étudiant.
ENONCE : RECEVOIR DOSSIER DE TYPE ty_étudiant IDENTIFIANT PAR étudiant
AVEC ETAT = reçu AVEC ECHEANCE = 1987/09/08
CONSTITUE
DE DOCUMENT DE TYPE ty_lettre INDENTIFIE PAR introduction
AVEC ETAT = reçu AVEC ECHEANCE = 1987/09/08,
DE COPIE 3 DE FORMULAIRE inscription IDENTIFIE PAR étudiant
AVEC ETAT = reçu AVEC ECHEANCE = 1987/09/08,
DE COPIE 4 DE FORMULAIRE inscription IDENTIFIE PAR étudiant
AVEC ETAT = reçu AVEC ECHEANCE = 1987/09/08
DECLENCHEE PAR SOUS-SCHEMA sous-schéma 2
DECLENCHE OPERATION PRIMITIVE op_prim 12

OPERATION PRIMITIVE op_prim 12
DESCRIPTION : la secrétaire feuillète le dossier.
ENONCE : FEUILLETER DOSSIER ty_étudiant IDENTIDIE PAR étudiant
AVEC ETAT = feuilleté
DECLENCHEE PAR OPERATION PRIMITIVE 11
DECLENCHE OPERATION PRIMITIVE op_prim 13

OPERATION PRIMITIVE op_prim 13
DESCRIPTION : la secrétaire vérifie la complétude du premier formulaire.
ENONCE : VERIFIER COPIE NUMERO 3 DE FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
AVEC ETAT = vérifié
AVEC RESULTAT VERIFICATION paramètre 1
DECLENCHEE PAR OPERATION PRIMITIVE 12
DECLENCHE OPERATION PRIMITIVE op_prim 14

OPERATION PRIMITIVE op_prim 14
DESCRIPTION : la secrétaire vérifie la complétude du deuxième formulaire.
ENONCE : VERIFIER COPIE NUMERO 4 DE FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
AVEC ETAT = vérifié
AVEC RESULTAT VERIFICATION paramètre 2
DECLENCHEE PAR OPERATION PRIMITIVE 13
DECLENCHE OPERATION PRIMITIVE op_prim 15

OPERATION PRIMITIVE op_prim 15
DESCRIPTION : la secrétaire replace le premier formulaire dans le dossier.
ENONCE : REPLACER COPIE NUMERO 3 DE FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DECLENCHEE PAR OPERATION PRIMITIVE 14
DECLENCHE OPERATION PRIMITIVE op_prim 16

OPERATION PRIMITIVE op_prim 16
DESCRIPTION : la secrétaire replace le deuxième formulaire dans le dossier.
ENONCE : REPLACER COPIE NUMERO 4 DE FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DECLENCHEE PAR OPERATION PRIMITIVE 15
DECLENCHE CONDITION condition 1

CONDITION condition 1
DESCRIPTION : on teste le résultat de la vérification du premier formulaire.
ENONCE : SI PARAMETRE paramètre 1 = "INCOMPLET"
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 16
SI VRAI DECLENCHE OPERATION PRIMITIVE op_prim 17
SI FAUX DECLENCHE CONDITION condition 2

CONDITION condition 2
DESCRIPTION : on teste le résultat de la vérification du deuxième formulaire.
ENONCE : SI PARAMETRE paramètre 2 = "INCOMPLET"
DECLENCHEE PAR CONDITION condition 1
SI VRAI DECLENCHE OPERATION PRIMITIVE op_prim 17
SI FAUX DECLENCHE OPERATION PRIMITIVE op_prim 20

OPERATION PRIMITIVE op_prim 17
DESCRIPTION : la secrétaire rédige une demande de renseignements complémentaires.
ENONCE : CREER DOCUMENT DE TYPE ty_lettre IDENTIFIE PAR renseignement
AVEC ETAT = crée AVEC ECHEANCE = 1987/09/12
DECLENCHEE PAR CONDITION condition 1,
PAR CONDITION condition 2
DECLENCHE OPERATION PRIMITIVE op_prim 18

OPERATION PRIMITIVE op_prim 18
DESCRIPTION : la secrétaire classe la demande de renseignement dans le dossier.
ENONCE : CLASSER DOCUMENT DE TYPE ty_lettre IDENTIFIE PAR renseignement
DANS DOSSIER DE TYPE ty_étudiant IDENTIFIE PAR étudiant
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 17
DECLENCHE OPERATION PRIMITIVE op_prim 19

OPERATION PRIMITIVE op_prim 19
DESCRIPTION : la secrétaire renvoit le dossier à l'étudiant.
ENONCE : COMMUNIQUER DOSSIER DE TYPE ty_étudiant IDENTIFIE PAR étudiant
AVEC ETAT = envoyé AVEC ECHEANCE = 1987/09/12
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 18
DECLENCHE FIN-SOUS-SCHEMA sous-schéma 2

OPERATION PRIMITIVE op_prim 20
DESCRIPTION : la secrétaire décide si l'étudiant doit ou non passer un examen d'entrée.
ENONCE : DECIDER A PARTIR DE DOSSIER ty_étudiant IDENTIFIE PAR étudiant
AVEC RESULTAT DECISION paramètre 3
DECLENCHEE PAR CONDITION condition 2
DECLENCHE CONDITION condition 3

CONDITION condition 3
DESCRIPTION : on teste le résultat de la décision.
ENONCE : SI PARAMETRE paramètre 3 = "EXAMEN ENTREE"
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 20
SI VRAI DECLENCHE OPERATION PRIMITIVE op_prim 21
SI FAUX DECLENCHE OPERATION PRIMITIVE op_prim 23

OPERATION PRIMITIVE op_prim 21
DESCRIPTION : la secrétaire rédige le document reprenant les examens d'entrée à présenter.
ENONCE : CREER DOCUMENT DE TYPE ty_lettre IDENTIFIE PAR examens
AVEC ETAT = crée AVEC ECHEANCE = 1987/09/20
DECLENCHEE PAR CONDITION condition 3
DECLENCHE OPERATION PRIMITIVE op_prim 22

OPERATION PRIMITIVE op_prim 22
DESCRIPTION : la secrétaire envoit le document à l'étudiant.
ENONCE : COMMUNIQUER DOCUMENT DE TYPE ty_lettre IDENTIFIE PAR examens
AVEC ETAT = envoyé AVEC ECHEANCE = 1987/09/20
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 21
DECLENCHE OPERATION PRIMITIVE 23

OPERATION PRIMITIVE op_prim 23
DESCRIPTION : la secrétaire classe le premier formulaire dans le fichier des demandes d'inscription.
ENONCE : CLASSER COPIE NUMERO 3 DE FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
DANS FICHIER inscriptions
DECLENCHEE PAR CONDITION condition 3,
PAR OPERATION PRIMITIVE op_prim 22
DECLENCHE OPERATION PRIMITIVE 24

OPERATION PRIMITIVE op_prim 24
DESCRIPTION : la secrétaire classe le dossier dans le fichier des étudiants.
ENONCE : CLASSER DOSSIER DE TYPE ty_étudiant IDENTIFIE PAR étudiant
DANS FICHIER étudiants
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 23
DECLENCHE FIN-SOUS-SCHEMA sous-schéma 2

1.3.3 Tâche 3 : Rentrée académique

TACHE rentrée académique
DESCRIPTION : la secrétaire envoit une copie du fichier d'inscriptions au secrétariat central.
DECLENCHE SOUS-SCHEMA sous-schéma 3

SOUS-SCHEMA sous-schéma 3
DESCRIPTION : la secrétaire envoit une copie du fichier d'inscriptions au secrétariat central.
DECLENCHE PAR TACHE rentrée académique
DECLENCHE OPERATION PRIMITIVE op_prim 25

OPERATION PRIMITIVE op_prim 25
DESCRIPTION : la secrétaire prépare un nouveau fichier d'inscriptions vide.
ENONCE : CREER FICHIER duplicata
AVEC ETAT = à envoyer AVEC ECHEANCE = 1987/09/26
AVEC ORDRE DE CLASSEMENT = ALPHABETIQUE
DECLENCHEE PAR SOUS-SCHEMA sous-schéma 3
DECLENCHE BOUCLE boucle 1

BOUCLE boucle 1
DESCRIPTION : pour chaque formulaire du fichier d'inscriptions, le dupliquer en un exemplaire.
ENONCE : POUR CHAQUE CONSTITUANT OIE DE FICHIER inscriptions
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 25
POUR CHAQUE OCCURENCE DE variable 1 DECLENCHE SOUS-SCHEMA sous-schéma 4
ENSUITE DECLENCHE OPERATION PRIMITIVE op_prim 27

SOUS-SCHEMA sous-schéma 4
DESCRIPTION : dupliquer un formulaire.
DECLENCHE PAR BOUCLE boucle 1
DECLENCHE OPERATION PRIMITIVE op_prim 26

OPERATION PRIMITIVE op_prim 26
DESCRIPTION : dupliquer un formulaire.
ENONCE : REPRODUIRE 1 FOIS A PARTIR DU NUMERO 5
VARIABLE variable 1
DECLENCHEE PAR SOUS-SCHEMA sous-schéma 4
DECLENCHE FIN-SOUS-SCHEMA sous-schéma 4

OPERATION PRIMITIVE op_prim 27
DESCRIPTION : la secrétaire replace tous les formulaires dans le fichier d'inscriptions.
ENONCE : REPLACER TOUS LES FORMULAIRES DE TYPE inscription
DECLENCHEE PAR BOUCLE boucle 1
DECLENCHE BOUCLE boucle 2

BOUCLE boucle 2
DESCRIPTION : pour chaque copie effectuée, la classer dans le nouveau fichier.
ENONCE : POUR CHAQUE FORMULAIRE
DECLENCHEE PAR OPERATION PRIMITIVE op_prim 27
POUR CHAQUE OCCURENCE DE variable 2 DECLENCHE SOUS-SCHEMA sous-schéma 5
ENSUITE DECLENCHE OPERATION PRIMITIVE op_prim 29

SOUS-SCHEMA sous-schéma 5
DESCRIPTION : vérifier que c'est un formulaire que l'on vient de dupliquer et le classer.
DECLENCHE PAR BOUCLE boucle 2
DECLENCHE CONDITION condition 4

CONDITION condition 4
DESCRIPTION : vérifier que c'est un formulaire du bon type.
ENONCE : SI TYPE DE VARIABLE variable 2 EST = "inscription"
DECLENCHEE PAR SOUS-SCHEMA sous-schéma 5
SI VRAI DECLENCHE CONDITION condition 5
SI FAUX DECLENCHE FIN-SOUS-SCHEMA sous-schéma 5

CONDITION condition 5
DESCRIPTION : vérifier que c'est un formulaire avec le bon numéro de copie.
ENONCE : SI NUM_COPIE DE VARIABLE variable 2 EST = "5"
DECLENCHEE PAR CONDITION condition 4
SI VRAI DECLENCHE OPERATION PRIMITIVE op_prim 28
SI FAUX DECLENCHE FIN-SOUS-SCHEMA sous-schéma 5

OPERATION PRIMITIVE op_prim 28
DESCRIPTION : classer ce formulaire dans le nouveau fichier.
ENONCE : CLASSER VARIABLE variable 2 DANS FICHIER duplicata
DECLENCHEE PAR CONDITION condition 5
DECLENCHE FIN-SOUS-SCHEMA sous-schéma 5

OPERATION PRIMITIVE op_prim 29
DESCRIPTION : la secrétaire envoit le fichier dupliqué au secrétariat central.
ENONCE : COMMUNIQUER FICHIER duplicata
AVEC ETAT = envoyé AVEC ECHEANCE 1987/09/20
DECLENCHEE PAR BOUCLE boucle 2
DECLENCHE FIN-SOUS-SCHEMA sous-schéma 3

1.4 Déclaration de l'environnement

Nous allons dans ce point décrire l'environnement de ces différentes tâches. Cette description se fera grâce au langage défini au point 4.6

```
CREER OBJET INFORMATIONNEL FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant.
CREER OBJET INFORMATIONNEL FICHIER contacts AVEC ORDRE DE CLASSEMENT = AUCUN.
CREER OBJET INFORMATIONNEL FICHIER inscriptions AVEC ORDRE DE CLASSEMENT = AUCUN.
CREER OBJET INFORMATIONNEL FICHIER étudiants AVEC ORDRE DE CLASSEMENT = AUCUN.
CREER OBJET INFORMATIONNEL PILE pile_in AVEC ORDRE DE CLASSEMENT = LIFO.
CREER OBJET INFORMATIONNEL PILE pile_out AVEC ORDRE DE CLASSEMENT = LIFO.

CREER OUTIL DE TRAVAIL TABLE DE TRAVAIL.
CREER OUTIL DE TRAVAIL PHOTOCOPIEUSE.

CREER OBJET INTERFACE TELEPHONE.
CREER OBJET INTERFACE BOITE IN.
CREER OBJET INTERFACE BOITE OUT.
CREER OBJET INTERFACE POUBELLE.

CREER OBJET RANGEMENT ARMOIRE armoire1.
CREER OBJET RANGEMENT ETAGERE étagère 1.
CREER OBJET RANGEMENT FARDE farde1.
CREER OBJET RANGEMENT TIROIR tiroir1.
CREER OBJET RANGEMENT TIROIR tiroir2.
CREER OBJET RANGEMENT TIROIR tiroir3.
CREER OBJET RANGEMENT BOITE boîte1.

CREER ASSOCIATION RANGEMENT ENTRE FORMULAIRE DE TYPE inscription IDENTIFIE PAR étudiant
                                   ET FARDE farde 1.
CREER ASSOCIATION RANGEMENT ENTRE FICHIER contacts ET TIROIR tiroir1.
CREER ASSOCIATION RANGEMENT ENTRE FICHIER inscriptions ET TIROIR tiroir2.
CREER ASSOCIATION RANGEMENT ENTRE FICHIER étudiants ET TIROIR tiroir3.

CREER ASSOCIATION INTERFACE ENTRE PILE pile_in ET BOITE IN.
CREER ASSOCIATION INTERFACE ENTRE PILE pile_out ET BOITE OUT.

CREER ASSOCIATION COMPOSITION ENTRE TIROIR tiroir1 ET ARMOIRE armoire1.
CREER ASSOCIATION COMPOSITION ENTRE TIROIR tiroir2 ET ARMOIRE armoire1.
CREER ASSOCIATION COMPOSITION ENTRE TIROIR tiroir3 ET ARMOIRE armoire1.
CREER ASSOCIATION COMPOSITION ENTRE FARDE farde1 ET BOITE boîte1.
CREER ASSOCIATION COMPOSITION ENTRE BOITE boîte1 ET ETAGERE étagère1.
```

1.5 Conclusion

Cette annexe nous a permis de tirer des conclusions quant aux possibilités et limites du langage dans le chapitre 8.

Annexe 2
Description des constantes et des structures

ANNEXE 2 : DESCRIPTION DES CONSTANTES ET DES STRUCTURES

Nous allons décrire dans cette annexe les constantes et les structures qui seront utilisées dans les spécifications externes des modules de l'architecture logique. Cette architecture a été décrite au chapitre 7. Les spécifications externes seront exposées à l'annexe 3.

2.1 Description des constantes et des structures communes à tous les modules

A) Définition des constantes

MX_ID_CONCEPT
Longueur maximale de l'identifiant d'un concept du langage. (20).

MX_ID_TACHE
Longueur maximale de l'identifiant d'une tâche. (20).

MX_ID_SOUS_SCHEMA
Longueur maximale de l'identifiant d'un sous-schéma. (20).

MX_ID_BOUCLE
Longueur maximale de l'identifiant d'une boucle. (20).

MX_ID_CONDITION
Longueur maximale de l'identifiant d'une condition. (20).

MX_ID_OPERATION
Longueur maximale de l'identifiant d'une opération primitive. (20).

MX_TY_OPERATION
Longueur maximale du type d'une opération primitive. (15).

MX_LG_DESCRIPTION
Longueur maximale de la description d'une opération primitive. (80).

MX_LG_ATTRIBUT
Longueur maximale d'un attribut d'un objet informationnel. (à l'exception de la description). (10).

MX_LG_CLASSE
Longueur maximale de la classe d'un objet informationnel. (10).

MX_LG_TYPE
Longueur maximale du type d'un objet informationnel. (10).

MX_LG_NOM
Longueur maximale du nom d'un objet informationnel. (10).

MX_LG_ETAT
Longueur maximale de l'état d'un objet informationnel. (10).

MX_LG_ECHEANCE
Longueur maximale de la date d'échéance d'un objet informationnel. (10). (format AAAA/MM/JJ).

MX_LG_ORDRE
Longueur maximale d'un ordre de classement d'un objet informationnel. (20).

MX_LG_ETAT_CONTENU
Longueur maximale de l'état du contenu d'un objet informationnel. (10)

MX_LG_PARAMETRE
Longueur maximale du nom d'un paramètre dans le système. (20).

MX_LG_VALEUR_PARAMETRE
Longueur maximale de la valeur d'un paramètre dans le système. (15).

MX_LG_VARIABLE
Longueur maximale du nom d'une variable dans le système. (20).

MX_OBJET_CLASSE
Longueur maximale de la classe d'un objet de rangement, d'interface ou d'un outil de travail. (20).

MX_OBJET_NOM
Longueur maximale du nom d'un objet de rangement, d'interface ou d'un outil de travail. (20).

MX_OR_CLASSE
Longueur maximale de la classe d'un objet de rangement. (10).

MX_OR_NOM
Longueur maximale du nom d'un objet de rangement. (15).

MX_OI_CLASSE
Longueur maximale de la classe d'un objet d'interface. (10).

MX_OI_NOM
Longueur maximale du nom d'un objet d'interface. (15).

MX_OT_CLASSE
Longueur maximale de la classe d'un outil de travail. (20).

MX_OT_NOM
Longueur maximale du nom d'un outil de travail. (20).

MX_LG_MODE
Longueur maximale du mode de réception ou de communication d'un objet informationnel. (5).

MX_LG_DESIR
Longueur maximale du désir de l'utilisateur pour sélectionner les constituants d'un objet informationnel structurant. (4).

MX_LG_OPERATEUR
Longueur maximale d'un opérateur dans une condition. (2).

MX_LG_CST
Longueur maximale d'une constante alphanumérique dans une condition. (20).

B) Définition des structures

• Structure ST_CREER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ordre_clas = string [MX_LG_ORDRE] (20).

• Structure ST_REPRODUIRE

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Nbre_copie = entier (2).
- Num_départ = entier (2).

• Structure ST_REPRODUIRE_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).

- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Nbre_copie = entier (2).
- Num_départ = entier (2).

• Structure ST_RECEVOIR

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ordre_clas = string [MX_LG_ORDRE] (20).
- Mode_reception = string [MX_LG_MODE] (5).
- Composants = pointeur vers une structure ST_LISTE_VALEUR_OI.

• Structure ST_COMMUNIQUER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Mode_communication = string [MX_LG_MODE] (5).

• Structure ST_COMMUNIQUER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Mode_communication = string [MX_LG_MODE] (5).

• Structure ST_CONSULTER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).

- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_CONSULTER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_FEUILLETER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ordre_feuilletage = string [MX_LG_ORDRE] (20).

• Structure ST_FEUILLETER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ordre_feuilletage = string [MX_LG_ORDRE] (20).

• Structure ST_ENLEVER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Nom_pile = string [MX_LG_NOM] (15).

• Structure ST_ENLEVER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Nom_pile = string [MX_LG_NOM] (15).

- Variable_constituant = string [MX_LG_VARIABLE] (20).

• Structure ST_MODIFIER

- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Nom_message = string [MX_LG_NOM] (15).
- Num_copie_message = entier (2).

• Structure ST_MODIFIER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Nom_message = string [MX_LG_NOM] (15).
- Num_copie_message = entier (2).

• Structure ST_COMPLETER

- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_COMPLETER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_VERIFIER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).

- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Nom_paramètre = string [MX_LG_PARAMETRE] (20).

• Structure ST_VERIFIER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Nom_paramètre = string [MX_LG_PARAMETRE] (20).

• Structure ST_DETRUIRE

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_DETRUIRE_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_ARCHIVER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_ARCHIVER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

• Structure ST_TRIER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ordre_tri = string [MX_LG_ORDRE] (20).

• Structure ST_TRIER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ordre_tri = string [MX_LG_ORDRE] (20).

• Structure ST_CLASSER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Dest_classe = string [MX_LG_CLASSE] (10).
- Dest_type = string [MX_LG_TYPE] (10).
- Dest_nom = string [MX_LG_NOM] (15).

• Structure ST_CLASSER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).

- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Dest_classe = string [MX_LG_CLASSE] (10).
- Dest_type = string [MX_LG_TYPE] (10).
- Dest_nom = string [MX_LG_NOM] (15).

• Structure ST_RANGER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Dest_classe = string [MX_OR_CLASSE] (10).
- Dest_nom = string [MX_OR_NOM] (15).

• Structure ST_RANGER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Dest_classe = string [MX_OR_CLASSE] (10).
- Dest_nom = string [MX_OR_NOM] (15).

• Structure ST_REPLACER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Gr_classe = string [MX_LG_CLASSE] (10).
- Gr_type = string [MX_LG_TYPE] (10).

• Structure ST_REPLACER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Gr_classe = string [MX_LG_CLASSE] (10).
- Gr_type = string [MX_LG_TYPE] (10).

• Structure ST_DECIDER

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Nom_paramètre = string [MX_LG_PARAMETRE] (20).

• Structure ST_DECIDER_VAR

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Nom_paramètre = string [MX_LG_PARAMETRE] (20).

• structure ST_OPERATION

- Type_opération = string [MX_TY_OPERATION] (15).
- Nom_opération = string [MX_ID_OPERATION] (20).
- Description = string [MX_LG_DESCRIPTION] (80).
- Information : champ variable composé de :

  ° Créer = structure ST_CREER.
  ° Reproduire = structure ST_REPRODUIRE.
  ° Recevoir = structure ST_RECEVOIR.
  ° Communiquer = structure ST_COMMUNIQUER.
  ° Consulter = structure ST_CONSULTER.
  ° Feuilleter = structure ST_FEUILLETER.
  ° Enlever = structure ST_ENLEVER.
  ° Modifier = structure ST_MODIFIER.
  ° Compléter = structure ST_COMPLETER.
  ° Vérifier = structure ST_VERIFIER.
  ° Détruire = structure ST_DETRUIRE.
  ° Archiver = structure ST_ARCHIVER.
  ° Trier = structure ST_TRIER.
  ° Classer = structure ST_CLASSER.
  ° Ranger = structure ST_RANGER.
  ° Replacer = structure ST_REPLACER.
  ° Décider = structure ST_DECIDER.

• structure ST_OPERATION_VAR

- Type_opération = string [MX_TY_OPERATION] (15).
- Nom_opération = string [MX_ID_OPERATION] (20).
- Description = string [MX_LG_DESCRIPTION] (80).
- Information : champ variable composé de :

  ° Créer = structure ST_CREER.
  ° Reproduire = structure ST_REPRODUIRE_VAR.
  ° Recevoir = structure ST_RECEVOIR.
  ° Communiquer = structure ST_COMMUNIQUER_VAR.
  ° Consulter = structure ST_CONSULTER_VAR.
  ° Feuilleter = structure ST_FEUILLETER_VAR.
  ° Enlever = structure ST_ENLEVER_VAR.
  ° Modifier = structure ST_MODIFIER_VAR.
  ° Compléter = structure ST_COMPLETER_VAR.
  ° Vérifier = structure ST_VERIFIER_VAR.
  ° Détruire = structure ST_DETRUIRE_VAR.
  ° Archiver = structure ST_ARCHIVER_VAR.
  ° Trier = structure ST_TRIER_VAR.
  ° Classer = structure ST_CLASSER_VAR.
  ° Ranger = structure ST_RANGER_VAR.
  ° Replacer = structure ST_REPLACER_VAR.
  ° Décider = structure ST_DECIDER_VAR.

• Structure ST_C_DECLENCHEUR

- Type_concept = entier (1). Ensemble des valeurs : {TACHE, SOUS_SCHEMA, BOUCLE, CONDITION, OPERATION_PRIMITIVE).
- Id_concept_déclencheur = string [MX_ID_CONCEPT] (20).
- Critère : champ variable composé de :

  ° Condition = entier (1). Ensemble des valeurs : {VRAI, FAUX).
  ° Boucle = entier (1). Ensemble des valeurs : {SOUS_SCHEMA, SUITE).

• Structure ST_C_DECLENCHE

- Type_concept = entier (1). Ensemble des valeurs : {SOUS_SCHEMA, BOUCLE, CONDITION, OPERATION_PRIMITIVE).
- Id_concept_déclenché = string [MX_ID_CONCEPT] (20).

• Structure ST_CRITERE_BOUCLE

- Nom_variable = string [MX_LG_VARIABLE] (20).
- Type_critère = entier (1). Ensemble des valeurs : (CR_CONSTITUANT, CR_CLASSE, CR_TOUS).
- Information : champ variable composé de :

  ° Critère_constituant : structure composée de :

  - Classe = string [MX_LG_CLASSE] (10).
  - Type = string [MX_LG_TYPE] (10).
  - Nom = string [MX_LG_NOM] (15).
  - Désir = string [MX_LG_DESIR] (4).
  - Variable = string [MX_LG_VARIABLE] (20).

  ° Critère_classe : structure composée de :

  - Classe = string [MX_LG_CLASSE] (10).

• Structure ST_CLAUSE_CONDITION

- Type_clause = entier (1). Ensemble des valeurs : (CL_ATTRIBUT, CL_APPARTENANCE, CL_PARAMETRE).
- Information : champ variable composé de :

  ° Clause_attribut : structure composée de :

  - Nom_attribut = string [MX_LG_ATTRIBUT] (10).
  - Cst_alpha_num = string [MX_LG_CST] (20).
  - Opérateur = string [MX_LG_OPERATEUR] (2).
  - Nom_variable = string [MX_LG_VARIABLE] (20).
  - Classe = string [MX_LG_CLASSE] (10).
  - Type = string [MX_LG_TYPE] (10).
  - Nom = string [MX_LG_NOM] (15).
  - Num_copie = entier (2).

  ° Clause_appartenance : structure composée de :

  - OI_nom_variable = string [MX_LG_VARIABLE] (20).
  - OI_classe = string [MX_LG_CLASSE] (10).
  - OI_type = string [MX_LG_TYPE] (10).
  - OI_nom = string [MX_LG_NOM] (15).
  - OI_num_copie = entier (2).
  - OIS_nom_variable = string [MX_LG_VARIABLE] (20).

- OIS_classe = string [MX_LG_CLASSE] (10).
- OIS_type = string [MX_LG_TYPE] (10).
- OIS_nom = string [MX_LG_NOM] (15).

° Clause_parametre : structure composée de :

- Nom_parametre = string [MX_LG_PARAMETRE] (20).
- Cst_alpha_num = string [MX_LG_CST] (20).
- Opérateur = string [MX_LG_OPERATEUR] (2).

• Structure ST_LISTE_IDENTIFIANT_OI

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Ptr_suivant = pointeur vers une structure ST_LISTE_IDENTIFIANT_OI.

• Structure ST_LISTE_VALEUR_OI

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).
- Ptr_suivant = pointeur vers une structure ST_LISTE_VALEUR_OI.

• Structure ST_IDENTIFIANT_OI

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).

• Structure ST_VALEUR_OI

- Classe = string [MX_LG_CLASSE] (10).
- Type = string [MX_LG_TYPE] (10).
- Nom = string [MX_LG_NOM] (15).
- Num_copie = entier (2).
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

- Ordre_clas = string [MX_LG_ORDRE] (20).
- Etat_contenu = string [MX_LG_ETAT_CONTENU] (10).

• Structure ST_COMP_RECEPTION

- Nom_operation = string [MX_ID_OPERATION] (20).
- Identifiant_oi = structure ST_IDENTIFIANT_OI.
- Etat = string [MX_LG_ETAT] (10).
- Echéance = string [MX_LG_ECHEANCE] (10).

2.2 Description des structures du module graphique

• Structure ST_SE_COORDONNEES

- X1 = entier (2).
- Y1 = entier (2).
- X2 = entier (2).
- Y2 = entier (2).

• Structure ST_SE_OBJET_REPLACER

- Identifiant = structure ST_IDENTIFIANT_OI.
- Destination = structure ST_SE_ORIGINE.

• Structure ST_SE_ORIGINE

- Type_objet_origine = entier (2). Domaine de valeurs : {TIROIR, ETAGERE, FARDE, BOITE, DOSSIER, FICHIER, PILE, TABLE_DE_TRAVAIL, PHOTOCOPIEUSE}.
- Coordonnees_objet = structure ST_SE_COORDONNEES. Si l'originr est un O.I. alors cette structure n'a pas de valeur.
- Ptr_suivant = pointeur vers une structure ST_SE_ORIGINE.

• Structure ST_SE_CREER

- Info_créer = structure ST_CREER.
- Origine = structure ST_SE_COORDONNEES.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_REPRODUIRE

- Info_reproduire = structure ST_REPRODUIRE.
- Origine = structure ST_SE_ORIGINE.

- Origine_copie = structure ST_SE_COORDONNEES.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_RECEVOIR

- Info_recevoir = structure ST_RECEVOIR.
- Origine_écrit = structure ST_SE_COORDONNEES.
- Destination_écrit = structure ST_SE_COORDONNEES.
- Origine_oral = structure ST_SE_COORDONNEES.
- Destination_oral = structure ST_SE_COORDONNEES.
- Lieu_téléphone = structure ST_SE_COORDONNEES.
- Longueur_fil = entier (2).

• Structure ST_SE_COMMUNIQUER

- Info_communiquer = structure ST_COMMUNIQUER.
- Origine_écrit = structure ST_SE_ORIGINE.
- Destination_écrit = structure ST_SE_COORDONNEES.
- Origine_oral = structure ST_SE_ORIGINE.
- Destination_écrit = structure ST_SE_COORDONNEES.

• Structure ST_SE_CONSULTER

- Info_consulter = structure ST_CONSULTER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_FEUILLETER

- Info_feuilleter = structure ST_FEUILLETER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.
- Liste_composant_trié = structure ST_LISTE_IDENTIFIANT_OI.

• Structure ST_SE_ENLEVER

- Info_enlever = structure ST_ENLEVER.
- Origine = structure ST_SE_COORDONNEES.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_MODIFIER

- Info_modifier = structure ST_MODIFIER.
- Origine_oi_modifié = structure ST_SE_ORIGINE.
- Origine_message = structure ST_SE_ORIGINE.

- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_COMPLETER

- Info_compléter = structure ST_COMPLETER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_VERIFIER

- Info_vérifier = structure ST_VERIFIER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_DETRUIRE

- Info_détruire = structure ST_DETRUIRE.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_ARCHIVER

- Info_archiver = structure ST_ARCHIVER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.

• Structure ST_SE_TRIER

- Info_trier = structure ST_TRIER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.
- Liste_composant_trié = structure ST_LISTE_IDENTIFIANT.

• Structure ST_SE_CLASSER

- Info_classer = structure ST_CLASSER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_ORIGINE.

• Structure ST_RANGER

- Info_ranger = structure ST_RANGER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_ORIGINE.

• Structure ST_SE_REPLACER

- Info_replacer = structure ST_REPLACER.
- Origine = structure ST_SE_COORDONNEES.
- Liste_objet = structure ST_SE_OBJET_REPLACER.

• Structure ST_SE_DECIDER

- Info_décider = structure ST_DECIDER.
- Origine = structure ST_SE_ORIGINE.
- Destination = structure ST_SE_COORDONNEES.

Annexe 3
Spécifications externes

ANNEXE 3 : SPECIFICATIONS EXTERNES

Nous allons spécifier dans cette annexe les primitives des modules de l'architecture logique. Cette architecture a été décrite au chapitre 7.

Remarques préliminaires

Les constantes et les structures qui sont utilisées dans les spécifications externes, ont été décrites à l'annexe 2.

Pour toutes les primitives, nous conviendrons que leurs arguments sont passés par adresse.

Après l'appel à une primitive d'un module, les résultats de celle-ci ne devront être pris en compte que si le code d'erreur n'indique aucune erreur.

Un code d'erreur dont la valeur est égale à OK indique qu'aucune erreur n'est survenue durant l'exécution d'une primitive.

Un code d'erreur dont la valeur est différente de OK indique qu'une erreur est survenue durant l'exécution d'une primitive.

Pour toutes les primitives, l'argument Code_retour est de type TY_CODE_RETOUR c-à-d entier (2).

Pour toutes les primitives, l'argument Code_erreur est de type TY_CODE_ERREUR c-à-d entier (2).

3.1 Module SIMULATION

• SIMULATION (Identifiant_tâche)

a) Spécification :

A partir de l'identifiant d'une tâche, cette primitive va simuler graphiquement cette tâche à l'écran en fonction de la description de la dynamique de cette tâche mémorisée dans la base de données du langage.

b) Entrée :

- Identifiant_tâche : identifiant de la tâche qu'il faut simuler.
  format : string [MX_ID_TACHE].

c) Sortie :

3.2 Module SEQUENCEUR

• SEQUENCEUR_TACHE (Identifiant_tâche, Opération_primitive, Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'une tâche, cette primitive va rechercher dans la base de données, la première opération primitive à simuler pour cette tâche. Ce module se charge de faire les initialisations nécessaires pour l'enregistrement de variables, paramètres, boucles et sous-schémas. Si une erreur survient dans le séquencement, cette primitive affiche un message d'erreur.

b) Entrée :

- Identifiant_tâche : identifiant de la tâche dont il faut rechercher la première opération primitive.
  format : string [MX_ID_TACHE].

c) Sortie :

- Opération_primitive : ensemble des informations concernant l'opération primitive à simuler.
  format : structure ST_OPERATION.
- Code_retour : OK
  FIN_TACHE : il n'y a plus d'opération primitive à simuler.
- Code_erreur : OK
  ERREUR : erreur dans le séquencement.

• SEQUENCEUR_OP_PRIM (Identifiant_op_prim, Opération_primitive, Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'une opération primitive, cette primitive va fournir l'opération primitive suivante à simuler. Si une erreur survient dans le séquencement, cette primitive affiche un message d'erreur.

b) Entrée :

- Identifiant_op_prim : identifiant de l'opération primitive dont il faut rechercher la suivante.
  format : string [MX_ID_OPERATION].

c) Sortie :

- Opération_primitive : ensemble des informations concernant l'opération primitive suivante à simuler.
  format : structure ST_OPERATION.
- Code_retour : OK
  FIN_TACHE : il n'y a plus d'opération primitive à simuler.
- Code_erreur : OK
  ERREUR : erreur dans le séquencement.

3.3 Module CONCEPT_SUIVANT

• CONCEPT_SUIVANT (Concept_déclencheur, Concept_déclenché,Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'un concept du langage, cette primitive fournira le concept déclenché par celui-ci. Dans le cas où le concept en entrée est une condition, il faut préciser quelle alternative on doit prendre en compte. Dans le cas où le concept en entrée est une boucle, il faut préciser si la boucle déclenche son sous-schéma ou l'instruction suivante.

b) Entrée :

- Concept_déclencheur : identifiant du concept déclencheur.
  format : structure ST_C_DECLENCHEUR.

c) Sortie :

- Concept_déclenché : identifiant du concept déclenché.
  format : structure ST_C_DECLENCHE.
- Code_retour : OK
  FIN_SOUS_SCHEMA : le concept déclenché est la fin d'un sous-schéma. L'identifiant du sous-schéma terminé se trouve dans le Concept_déclenché.
- Code_erreur : OK

3.4 Module SOUS_SCHEMA

• SOUS_SCHEMA (Identifiant_sous_schéma, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet d'enregistrer dans le système l'identifiant d'un sous-schéma déclenché.

Parmi tous les sous-schémas déclenchés et enregistrés dans le système à un instant donné, le dernier sous-schéma enregistré sera appelé le *sous-schéma actif*. A tout instant, dans le système, il y a toujours un et un seul sous-schéma actif.

b) Entrée :

- Identifiant_sous_schéma : identifiant du sous-schéma qui vient d'être déclenché.
  format : string [MX_ID_SOUS_SCHEMA].

c) Sortie :

- Code_retour : OK
- Code_erreur : OK
  ERREUR_SOUS_SCHEMA : le sous-schéma a été redéclenché sans avoir été terminé.
  ENREG_SOUS_SCHEMA : erreur du système dans l'enregistrement du sous-schéma dans le système.

3.5 Module FIN_SOUS_SCHEMA

• FIN_SOUS_SCHEMA (Identifiant_sous_schéma, Code_retour,Code_erreur)

a) Spécification :

Cette primitive va enregistrer dans le système le fait que le sous-schéma actif est terminé. Deux cas sont alors possibles. Soit il existe un sous-schéma qui a été déclenché avant celui qui vient de se terminer et qui devient le sous-schéma actif. Soit, il n'y a pas de sous-schéma déclenché avant celui qui vient de se terminer et dans ce cas, la tâche est terminée.

b) Entrée :

- Identifiant_sous_schéma : identifiant du sous-schéma terminé.
  format : string [MX_ID_SOUS_SCHEMA].

c) Sortie :

- Code_retour : FIN_TACHE : il n'y a plus de sous-schéma actif dans le système.
  BOUCLE_NEXT : il y a un nouveau sous-schéma actif dans le système

- Code_erreur : OK
  ERREUR_FIN_SOUS_SCHEMA : le sous-schéma terminé n'est pas celui qui est actif.
  SUPPR_SOUS_SCHEMA : erreur du système dans la suppression du sous-schéma.

3.6 Module FIN_TACHE

• FIN_TACHE (Code_erreur)

a) Spécification :

Cette primitive supprime toutes les variables et tous les paramètres enregistrés dans le système au cours de la tâche.

b) Entrée :

c) Sortie :

- Code_erreur : OK
  SUPPR_VARIABLE : erreur du système dans la suppression des variables.
  SUPPR_PARAMETRE : erreur du système dans la suppression des paramètres.

3.7 Module BOUCLE

• BOUCLE_FIRST (Identifiant_boucle, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet d'enregistrer dans le système l'identifiant d'une boucle qui commence son exécution. De plus, cette primitive va sélectionner l'ensemble des objets informationnels qui seront à traiter par cette boucle et l'y associer. Il enregistre également dans la variable associée à la boucle, le premier objet informationnel à traiter, pour autant que l'ensemble ne soit pas vide.

b) Entrée :

- Identifiant_boucle : identifiant de la boucle qui commence son exécution.
  format : string [MX_ID_BOUCLE].

c) Sortie :

- Code_retour : OK
  FIN_BOUCLE : l'ensemble des objets à traiter est vide.
- Code_erreur : OK
  ERREUR_BOUCLE : la boucle a déjà été redéclenchée sans avoir été terminée.
  OIS_BOUCLE : l'objet informationnel structurant dont on doit rechercher les constituants n'existe pas.
  VAR_EXISTE_PAS : la variable associée à la boucle n'existe pas.
  ENREG_VARIABLE : erreur du système dans l'enregistrement d'une variable.
  ENREG_BOUCLE : erreur du système dans l'enregistrement d'une boucle .
  ENREG_OBJET : erreur du système dans l'association d'un objet à une boucle.

• BOUCLE_NEXT (Identifiant_boucle, Code_retour, Code_erreur)

a) Spécification :

Cette primitive va assigner la variable associée à la la boucle au moyen de l'objet suivant à traiter. Il se peut qu'aucun objet suivant ne soit à traiter et que donc la boucle actuelle soit terminée. Dans ce cas, on enregistre dans le système le fait que cette boucle est terminée.

b) Entrée :

c) Sortie :

- Identifiant_boucle : identifiant de la boucle qui est en cours d'exécution
  format : string [MX_ID_BOUCLE].
- Code_retour : OK
  FIN_BOUCLE : la boucle est terminée.
- Code_erreur : OK
  LECTURE_OBJET : erreur du système dans la lecture d'un objet associé à une boucle.
  SUPPR_BOUCLE : erreur du système dans la suppression d'une boucle.
  SUPPR_OBJET : erreur du système dans la suppression d'un objet.

3.8 Module CONDITION

• CONDITION (Identifiant_condition, Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'une condition, cette primitive va évaluer la condition. Dans le cas où aucune erreur n'est intervenue, le résultat de cette évaluation est soit vrai soit faux.

b) Entrée :

- Identifiant_condition : identifiant de la condition qui doit être évaluée.
  format : string [MX_ID_CONDITION].

c) Sortie :

- Code_retour : VRAI : le résultat de l'évaluation est *vrai*
  FAUX : le résultat de l'evaluation est *faux*.
- Code_erreur : OK
  - OIE_CONDITION : un objet informationnel élémentaire utilisé dans la clause de la condition, n'existe pas.
  - OIS_CONDITION : un objet informationnel structurant utilisé dans la clause de la condition n'existe pas.
  - VAR_EXISTE_PAS : une variable utilisée dans la clause de la condition n'existe pas
  - PAR_EXISTE_PAS : un paramètre utilisé dans la clause de la condition n'existe pas.
  - CLAUSE_ATTRIBUT : la condition teste un attribut invalide étant donné la classe de l'objet informationnel.
  - CLAUSE_APPARTENACE : la condition teste l'appartenance d'un objet informationnel invalide étant donné sa classe.

3.9 Module OPERATION_PRIMITIVE

• OPERATION_PRIMITIVE (Identifiant_opération, Information_opération, Code_ retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'une opération primitive, cette primitive va rechercher l'information détaillée concernant cette opération dans la base de données du langage. Cette primitive est également chargée de supprimer la notion de variable en la remplacant par sa valeur.

b) Entrée :

- Identifiant_opération : identifiant de l'opération dont il faut rechercher l'information.
  format : string [MX_ID_OPERATION].

c) Sortie :

- Information_opération : ensemble des informations concernant l'opération primitive.
  format : structure ST_OPERATION.
- Code_retour : OK
- Code_erreur : OK
  - ENREG_VARIABLE : erreur du système dans l'enregistrement d'une variable.
  - VAR_EXISTE_PAS : une variable qui est utilisée dans l'énoncé de l'opération primitive n'existe pas.
  - OI_ENLEVER : la pile utilisée dans une opération *enlever* n'existe pas dans la base de données.
  - CLASSE_ENLEVER : la variable utilisée dans une opération *enlever* ne référence pas une pile.
  - CLASSE_MODIFIER : la variable utilisée dans une opération *modifier* ne référence pas un document.
  - CLASSE_COMPLETER : la variable utilisée dans une opération *completer* ne référence pas un formulaire.

3.10 Module VARIABLE

• INITIALISATION_VARIABLE ().

a) Spécification :

Cette primitive initialise le système et prépare celui-ci à l'enregistrement de variables.

b) Entrée :

c) Sortie :

• ASSIGNATION_VARIABLE (Nom_variable, Valeur_variable, Code_retour, Code_erreur)

a) Spécification :

A partir d'un nom de variable et d'une valeur, cette primitive va assigner cette variable au moyen de cette valeur. Si la variable existe déjà dans le système, son ancienne valeur est écrasée par la nouvelle valeur. Si la variable n'existe pas encore alors elle est créée.

b) Entrée :

- Nom_variable : identifiant de la variable que l'on doit assigner.
  format : string [MX_LG_VARIABLE].
- Valeur_variable : valeur avec laquelle on doit assigner la variable.
  format : structure ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la création d'une variable.

• LECTURE_VARIABLE (Nom_variable, Valeur_variable, Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'une variable, cette primitive va retourner la valeur qui est affectée à cette variable.

b) Entrée :

- Nom_variable : identifiant de la variable dont on veut connaitre la valeur.
  format : string [MX_LG_VARIABLE].

c) Sortie :

- Valeur_variable : valeur qui est associée à la variable.
  format : structure ST_IDENTIFIANT_OI.
- Code_retour : OK
- Code_erreur : OK
  VAR_EXISTE_PAS : la variable spécifiée n'existe pas.

• SUPPRESSION_VARIABLE (Code_erreur)

a) Spécification :

Cette primitive va supprimer toutes les variables enregistrées dans le système.

b) Entrée :

c) Sortie :

- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la suppression des variables.

3.11 Module PARAMETRE

• INITIALISATION_PARAMETRE ().

a) Spécification :

Cette primitive initialise le système et prépare celui-ci à l'enregistrement de parametres.

b) Entrée :

c) Sortie :

• ASSIGNATION_PARAMETRE (Nom_paramètre, Valeur_paramètre, Code_retour, Code_erreur)

a) Spécification :

A partir d'un nom de variable et d'une valeur, cette primitive va assigner ce paramètre au moyen de cette valeur. Si le paramètre existe déjà dans le système, son ancienne valeur est écrasée par la nouvelle valeur. Si le paramètre n'existe pas encore alors il est créé.

b) Entrée :

- Nom_paramètre : identifiant du paramètre que l'on doit assigner.
  format : string [MX_LG_PARAMETRE]
- Valeur_paramètre : valeur avec laquelle on doit assigner le paramètre.
  format : string [MX_LG_VALEUR_PARAMETRE].

c) Sortie :

- Code_retour : OK
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la création d'un paramètre.

• LECTURE_PARAMETRE (Nom_paramètre, Valeur_paramètre, Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'un paramètre, cette primitive va retourner la valeur qui est affectée à ce paramètre.

b) Entrée :

- Nom_paramètre : identifiant du paramètre dont on veut connaitre la valeur.
  format : string [MX_LG_PARAMETRE].

c) Sortie :

- Valeur_paramètre : valeur qui est associée au paramètre.
  format : string [MX_LG_VALEUR_PARAMETRE].
- Code_retour : OK
- Code_erreur : OK
  PAR_EXISTE_PAS : le paramètre spécifié n'existe pas.

• SUPPRESSION_PARAMETRE (Code_erreur)

a) Spécification :

Cette primitive va supprimer tous les paramètres enregistrés dans le système.

b) Entrée :

c) Sortie :

- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la suppression des paramètres.

3.12 Module MEMORISATION_SOUS_SCHEMA

• INITIALISATION_SOUS_SCHEMA ()

a) Spécification :

Cette primitive initialise le système et prépare celui-ci à l'enregistrement de sous-schémas.

b) Entrée :

c) Sortie :

• ENREGISTREMENT_SOUS_SCHEMA (Identifiant_sous_schéma, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet d'enregistrer dans le système l'identifiant d'un sous-schéma.

b) Entrée :

- Identifiant_sous_schéma : identifiant du sous-schéma qu'il faut mémoriser.
  format : string [MX_ID_SOUS_SCHEMA]

c) Sortie

- Code_retour : OK
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans l'enregistrement d'un sous-schéma.

• EXISTENCE_SOUS_SCHEMA (Identifiant_sous_schéma, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet de voir si un sous-schéma spécifié a été enregistré dans le système.

b) Entrée :

- Identifiant_sous_schéma : identifiant du sous-schéma dont l'existence doit être testée.
  format : string [MX_ID_SOUS_SCHEMA]

c) Sortie :

- Code_retour : PRESENT : le sous-schéma spécifié existe dans le système.
  ABSENT : le sous-schéma spécifié n'existe pas dans le système.
- Code_erreur : OK

• SUPPRESSION_SOUS_SCHEMA (Identifiant_sous_schéma, Code_retour, Code_erreur)

a) Spécification :

Cette primitive supprime le dernier sous-schéma mémorisé dans le système. Au moins un sous-schéma doit être actif avant l'appel de cette primitive.

b) Entrée :

c) Sortie :

- Identifiant_sous_schéma : identifiant du sous-schéma qui a été supprimé.
  format : string [MX_ID_SOUS_SCHEMA].

- Code_retour : OK
  DERNIER : le sous-schéma supprimé est le dernier qui était mémorisé dans le système.
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la suppression d'un sous-schéma.

3.13 Module MEMORISATION_BOUCLE

• INITIALISATION_BOUCLE ()

a) Spécification :

Cette primitive initialise le système et prépare celui-ci à l'enregistrement de boucles.

b) Entrée :

c) Sortie :

• ENREGISTREMENT_BOUCLE (Identifiant_boucle, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet d'enregistrer dans le système l'identifiant d'une boucle.

b) Entrée :

- Identifiant_boucle : identifiant de la boucle qu'il faut mémoriser.
  format : string [MX_ID_BOUCLE]

c) Sortie

- Code_retour : OK
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans l'enregistrement d'une boucle.

• EXISTENCE_BOUCLE (Identifiant_boucle, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet de voir si une boucle spécifiée a été enregistré dans le système.

b) Entrée :

- Identifiant_boucle : identifiant de la boucle dont l'existence doit être testée.
  format : string [MX_ID_BOUCLE].

c) Sortie :

- Code_retour : PRESENT : la boucle spécifiée existe dans le système
  ABSENT : la boucle spécifiée n'existe pas dans le système
- Code_erreur : OK

• SUPPRESSION_BOUCLE (Identifiant_boucle, Code_retour, Code_erreur)

a) Spécification :

Cette primitive supprime la dernière boucle mémorisé dans le système. Au moins une boucle doit être active avant l'appel de cette primitive. De plus, pour cette boucle, il ne doit plus exister d'objets informationnels à traiter.

b) Entrée :

c) Sortie :

- Identifiant_boucle : identifiant de la boucle à supprimer.
  format : string [MX_ID_BOUCLE]
- Code_retour : OK
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la suppression d'une boucle.

• AJOUTER_OCCURRENCE_BOUCLE (Identifiant_OI, Code_retour, Code_erreur)

a) Spécification :

A partir de l'identifiant d'un objet informationnel, cette primitive ajoute celui-ci à l'ensemble des objets informationnels associé à la dernière boucle mémorisée dans le système.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel qu'il faut ajouter à l'ensemble associé à la dernière boucle.
  format : structure ST_IDENTIFIANT_OI.

c) Sortie

- Code_retour : OK
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans l'ajout d'un objet.

• LIRE_OCCURRENCE_BOUCLE (Identifiant_boucle, Identifiant_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet de lire l'identifiant d'un objet informationnel qui est associé à la dernière boucle mémorisée dans le système. Il se peut qu'aucun objet ne soit associé à cette boucle. Dans le cas où il existe, il est supprimé de l'ensemble des objets associé à la dernière boucle. Cette primitive est assurée qu'au moins une boucle est enregistrée dans le système.

b) Entrée :

c) Sortie :

- Identifiant_boucle : identifiant de la dernière boucle mémorisée dans le système.
  format : string [MX_ID_BOUCLE]
- Identifiant_OI : identifiant d'un objet informationnel qui est associé à la dernière boucle mémorisée dans le système.
  format : structure ST_IDENTIFIANT_OI.
- Code_retour : OK
  VIDE : il n'y a plus d'objet informationnel associé à la dernière boucle mémorisée dans le système.
- Code_erreur : OK
  ERREUR_SYSTEME : erreur du système dans la suppression d'un objet.

3.14 Module CONTROLES_A_PRIORI

• CONTROLES_A_PRIORI (Information_op_prim, Code_retour, Code_erreur)

a) Spécification :

A partir des informations concernant une opération primitive, cette primitive va vérifier que l'exécution de cette opération est valide par rapport aux informations mémorisées dans la base de données informationnelle. Si l'opération est invalide, cette primitive affiche un message d'erreur.

b) Entrée :

- Information_op_prim : informations relatives à l'opération primitive que l'on veut vérifier.
  format : structure ST_OPERATION

c) Sortie :

- Code_retour : OK
- Code_erreur : OK
  ERREUR : la primitive a détecté une erreur et a affiché un message d'erreur.

3.15 GRAPHIQUE

• INITIALISATION_GRAPHIQUE (Code_erreur)

a) Spécification :

Cette primitive initialise le système et prépare celui-ci au graphisme.

b) Entrée :

c) Sortie :

- Code_erreur : OK
  ERREUR_GRAPHISME : erreur dans l'initialisation du graphisme.

• TERMINAISON_GRAPHIQUE (Code_erreur)

a) Spécification :

Cette primitive permet de clôturer le graphisme.

b) Entrée :

c) Sortie :

- Code_erreur : OK
  ERREUR_GRAPHISME : erreur dans la terminaison du graphisme.

• GRAPHIQUE (Information_op_prim, Code_retour, Code_erreur)

a) Spécification :

A partir de l'information concernant une opération primitive, ce module doit préparer et rechercher les informations nécessaires à l'animation graphique. Si cette primitive détecte une erreur, elle affiche un message d'erreur.

b) Entrée :

- Information_op_prim : informations relatives à l'opération à simuler.
  format : structure ST_OPERATION

c) Sortie :

- Code_retour : OK
- Code_erreur : OK
  ERREUR : la primitive a détecté une erreur et a affiché un message d'erreur.

3.16 Module MANIPULATION BD

• MANIPULATION_BD (Information_op_prim, Code_retour, Code_erreur)

a) Spécification :

A partir des informations concernant une opération primitive, ce module va apporter les modifications nécessaires à la base de données informationnelle de telle manière que sa cohérence sémantique soit respectée après exécution de cette opération primitive. Cette primitive est assurée qu'il n'existe plus aucune erreur soit dans l'information transmise, soit dans la validité des manipulations demandées sur l'information.

b) Entrée :

- Information_op_prim : informations nécessaires pour la mise à jour de la base de données.
  format : structure ST_OPERATION

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

3.17 Module MESSAGE_ERREUR

• MESSAGE_ERREUR (Numéro)

a) Spécification :

A partir du numéro d'un message d'erreur, ce module va afficher à l'écran le message correspondant à ce numéro.

b) Entrée :

- Numéro : numéro du message d'erreur à afficher.
  format : entier (2)

Le numéro peut prendre les valeurs suivantes :

ERREUR_BD_OUVERTURE
Une erreur est survenue durant l'ouverture de la base de données du langage ou de la base de données informationnelle.

ERREUR_BD_FERMETURE
Une erreur est survenue durant la fermeture de la base de données du langage ou de la base de données informationnelle.

ERREUR_GRAPHISME
Une erreur est survenue durant l'initialisation ou la terminaison du graphisme.

ERREUR_SOUS_SCHEMA
Un sous-schéma a été redéclenché sans avoir été terminé.

ERREUR_FIN_SOUS_SCHEMA
Un sous-schéma a été terminé alors qu'il n'était pas le sous-schéma actif.

ERREUR_BOUCLE
Une boucle a été redéclenchée sans avoir été terminée.

ENREG_SOUS_SCHEMA
Erreur du système dans l'enregistrement d'un sous-schéma

ENREG_BOUCLE
Erreur du système dans l'enregistrement d'une boucle.

ENREG_OBJET
Erreur du système dans l'enregistrement d'un objet informationnel à associer à une boucle

ENREG_VARIABLE
Erreur du système dans l'enregistrement d'une variable.

SUPPR_SOUS_SCHEMA
Erreur du système dans l'enregistrement d'un paramètre.

SUPPR_BOUCLE
Erreur du système dans la suppression d'une boucle.

SUPPR_OBJET
Erreur du système dans la suppression d'un objet associé à une boucle.

SUPPR_VARIABLE
Erreur du système dans la suppression des variables.

SUPPR_PARAMETRE
Erreur du système dans la suppression des paramètres

LECTURE_OBJET
Erreur du système dans la lecture d'un objet associé à une boucle

CLAUSE_ATTRIBUT
Une condition teste un attribut invalide étant donné la classe de l'objet informationnel.

CLAUSE_APPARTENANCE
Une condition teste l'appartenance d'un objet informationnel invalide étant donné sa classe.

OIE_CONDITION
Un objet informationnel élémentaire utilisé dans une clause d'une condition n'existe pas.

OIS_CONDITION
Un objet informationnel structurant utilisé dans une clause d'une condition n'existe pas.

OIS_BOUCLE
Un objet informationnel structurant dont on doit rechercher les constituants n'existe pas.

OI_CREER
Un objet informationnel que l'on doit *créer* existe déjà.

OIE_REPRODUIRE
Un objet informationnel élémentaire que l'on doit *reproduire* n'existe pas.

OI_RECEVOIR
Un objet informationnel que l'on doit *recevoir* ou l'un de ses composants existe déjà.

OI_COMMUNIQUER
Un objet informationnel que l'on doit *communiquer* n'existe pas.

OIE_CONSULTER
Un objet informationnel que l'on doit *consulter* n'existe pas.

OIS_FEUILLETER
Un objet informationnel que l'on doit *feuilleter* n'existe pas.

OI_ENLEVER
La pile dont on doit *enlever* le premier constituant n'existe pas ou est vide.

OI_MODIFIER
Un document que l'on doit *mofifier* n'existe pas ou le message brouillon à partir duquel on doit modifier n'existe pas.

OIE_COMPLETER
Un formulaire que l'on doit *compléter* n'existe pas.

OI_VERIFIER
Un dossier ou un formulaire que l'on doit *vérifier* n'existe pas.

OI_DETRUIRE
Un objet informationnel que l'on doit *détruire* n'existe pas.

OI_ARCHIVER
Un objet informationnel que l'on doit *archiver* n'existe pas.

OIS_TRIER
Un objet informationnel structurant que l'on doit *trier* n'existe pas.

OI_CLASSER
Un objet informationnel que l'on doit *classer* n'existe pas ou l'O.I.S. est un constituant de l'objet informationnel à *classer*.

OI_RANGER
Un objet informationnel que l'on doit *ranger* n'existe pas.

OI_REPLACER
Un objet informationnel que l'on doit *replacer* n'existe pas.

OI_DECIDER
Un objet informationnel sur base duquel on doit *décider*, n'existe pas.

VAR_EXISTE_PAS
Une variable spécifiée dans une boucle, une condition ou une opération primitive n'existe pas.

PAR_EXISTE_PAS
Un paramètre spécifié dans une condition n'existe pas.

PHOTOCOPIEUSE_EXISTE_PAS
Il n'existe pas de photocopieuse dans le système et donc l'opération *reproduire* est impossible.

BOITE_IN_EXISTE_PAS
Il n'existe pas de boîte IN dans le système et donc l'opération *recevoir* est impossible.

BOITE_OUT_EXISTE_PAS
Il n'existe pas de boîte OUT dans le système et donc l'opération *communiquer* est impossible.

BOITE_ARCHIVE_EXISTE_PAS
Il n'existe pas de boîte ARCHIVE dans le système et donc l'opération *archiver* est impossible.

TELEPHONE_EXISTE_PAS
Il n'existe pas de téléphone dans le système et donc l'opération de *communiquer* ou *recevoir* est impossible.

POUBELLE_EXISTE_PAS
Il n'existe pas de poubelle dans le système et donc l'opération de *détruire* est impossible.

CLASSEMENT_EXISTE_PAS
Un objet informationnel dans lequel on doit *classer* n'existe pas.

RANGEMENT_EXISTE_PAS
Un objet de rangement dans lequel on doit *ranger* n'existe pas.

PAS_SUR_OUTIL_TRAVAIL
Il existe une opération *replacer* mais l'objet à replacer ne se trouve pas sur un outil de travail.

NUM_DEPART_INCORRECT
Dans une opération *reproduire* , le numéro de départ spécifié n'est pas supérieur à tous les numéros de copie de l'original.

ERREUR_FICHIER
Un objet informationnel que l'on doit *classer* dans un fichier n'est pas compatible avec les autres objets informationnels se trouvant déjà dans celui-ci.

CLASSE_REPRODUIRE
Un objet informationnel que l'on doit *reproduire* est invalide étant donné sa classe.

CLASSE_COMMUNIQUER
Un objet informationnel que l'on doit *communiquer* est invalide étant donné sa classe.

CLASSE_CONSULTER
Un objet informationnel que l'on doit *consulter* est invalide étant donné sa classe.

CLASSE_FEUILLETER
Un objet informationnel que l'on doit *feuilleter* est invalide étant donné sa classe.

CLASSE_ENLEVER
Un objet informationnel dont on doit *enlever* le premier constituant n'est pas une pile.

CLASSE_MODIFIER
Un objet informationnel que l'on doit *modifier* est invalide étant donné sa classe.

CLASSE_COMPLETER
Un objet informationnel que l'on doit *compléter* est invalide étant donné sa classe.

CLASSE_VERIFIER
Un objet informationnel que l'on doit *vérifier* est invalide étant donné sa classe.

CLASSE_DETRUIRE
Un objet informationnel que l'on doit *détruire* est la pile qui est associée soit à la boîte IN soit à la boîte OUT.

CLASSE_ARCHIVER
Un objet informationnel que l'on doit *archiver* est invalide étant donné sa classe.

CLASSE_TRIER

Un objet informationnel que l'on doit *trier* est invalide étant donné sa classe.

CLASSE_CLASSER

Un objet informationnel que l'on doit *classer* est invalide étant donné sa classe.

CLASSE_DECIDER

Un objet informationnel sur base duquel on doit *décider* est invalide étant donné sa classe.

c) Sortie :

3.18 Module BASE_DE_DONNEES

• OUVERTURE_BD (Code_erreur)

a) Spécification :

Cette primitive ouvre la base de données du langage et la base de données informationnelle

b) Entrée :

c) Sortie :

- Code_erreur : OK
  ERREUR_BD_OUVERTURE : erreur dans l'ouverture d'une des deux bases de données.

• FERMETURE_BD (Code_erreur)

a) Spécification :

Cette primitive ferme la base de données du langage et la base de données informationnelle

b) Entrée :

c) Sortie :

- Code_erreur : OK
  ERREUR_BD_FERMETURE : erreur dans la fermeture d'une des deux bases de données.

• CRITERE_BOUCLE (Identifiant_boucle, Info_critère, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour objectif, à partir de l'identifiant d'une boucle, de retrouver les informations concernant le critère de cette boucle. Cette fonction est assurée que la boucle existe.

b) Entrée :

- Identifiant_boucle : identifiant de la boucle pour laquelle il faut retrouver le critère utilisé.
  format : string [MX_ID_BOUCLE]

c) Sortie :

- info_critère : champ variable contenant les informations relatives au critère de la boucle.
  format : structure ST_CRITERE_BOUCLE
- Code_retour : OK
  TROISIEME_CRITERE : il n'y a pas de critère associé à la boucle et donc il faut appliquer le troisième type de critère.
- Code_erreur : OK

• CLAUSE_CONDITION (Identifiant_condition, Info_clause, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour objectif, à partir de l'identifiant d'une condition, de retrouver les informations concernant la clause de la condition. Cette fonction est assurée que la condition existe et qu'une clause lui est associée.

b) Entrée :

- Identifiant_condition : identifiant de la condition pour laquelle il faut retrouver la clause utilisée.
  format : string [MX_ID_CONDITION].

c) Sortie :

- Info_clause : champ variable contenant les informations relatives à la clause de la condition.
  format : structure ST_CLAUSE_CONDITION.
- Code_retour : OK
- Code_erreur : OK

• SOUS_TYPE_OPERATION_PRIMITIVE (Identifiant_operation, Info_opération, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour objectif, à partir de l'identifiant d'une opération primitive, de retrouver l'information concernant le sous-type de cette opération. Cette fonction est assurée que cette opération primitive existe et qu'un sous-type lui est associé.

b) Entrée :

- Identifiant_opération : identifiant de l'opération primitive pour laquelle il faut retrouver l'information du sous-type.
  format : string [MX_ID_OPERATION]

c) Sortie :

- Info_opération : champ variable contenant les informations relatives au sous-type de l'opération primitive.
  format : structure ST_OPERATION_VAR
- Code_retour : OK
- Code_erreur : OK

• TACHE_CONCEPT_SUIVANT (Identifiant_tâche, Concept_déclenché, Code_retour, Code_Erreur)

a) Spécification :

A partir de l'identifiant d'une tâche, cette primitive fournira le sous-schéma déclenché par cette même tâche.

b) Entrée :

- Identifiant_tâche : identifiant de la tâche dont il faut rechercher le concept déclenché.
  format : string [MX_ID_TACHE].

c) Sortie :

- Concept_déclenché : identifiant du concept déclenché.
  format : structure ST_C_DECLENCHE.
- Code_retour : OK
- Code_erreur : OK

• SOUS_SCHEMA_CONCEPT_SUIVANT (Identifiant_sous_schéma, Concept_déclenché, Code_retour, Code_Erreur)

a) Spécification :

A partir de l'identifiant d'un sous-schéma, cette primitive fournira le concept déclenché par ce même sous-schéma.

b) Entrée :

- Identifiant_sous-schéma : identifiant du sous-schéma dont il faut rechercher le concept déclenché.
  format : string [MX_ID_SOUS_SCHEMA].

c) Sortie :

- Concept_déclenché : identifiant du concept déclenché.
  format : structure ST_C_DECLENCHE.
- Code_retour : OK
- Code_erreur : OK

• BOUCLE_CONCEPT_SUIVANT (Identifiant_boucle, Critere, Concept_déclenché, Code_retour, Code_Erreur)

a) Spécification :

A partir de l'identifiant d'une boucle, cette primitive fournira le concept déclenché par cette même boucle en fonction du critère spécifié. Ce critère indique s'il faut prendre en compte le sous-schéma déclenché par cette boucle ou s'il faut prendre en compte le concept suivant.

b) Entrée :

- Identifiant_boucle : identifiant de la boucle dont il faut rechercher le concept déclenché.
  format : string [MX_ID_BOUCLE].
- Critère : critère associé à la recherche.
  format : entier(1). Ensemble des valeurs :
  {SOUS_SCHEMA, SUITE}

c) Sortie :

- Concept_déclenché : identifiant du concept déclenché.
  format : structure ST_C_DECLENCHE.
- Code_retour : OK
  FIN_SOUS_SCHEMA : le concept déclenché est la fin d'un sous-schéma. L'identifiant du sous-schéma terminé se trouve dans Concept_déclenché.
- Code_erreur : OK

• CONDITION_CONCEPT_SUIVANT (Identifiant_condition, Critere, Concept_déclenché, Code_retour, Code_Erreur)

a) Spécification :

A partir de l'identifiant d'une condition, cette primitive fournira le concept déclenché par cette même condition en fonction du critère spécifié. Ce critère indique quelle branche de la condition il faut prendre en compte.

b) Entrée :

- Identifiant_condition : identifiant de la condition dont il faut rechercher le concept déclenché.
  format : string [MX_ID_CONDITION].
- Critère : critère associé à la recherche.
  format : entier (1). Ensemble des valeurs : {VRAI, FAUX}

c) Sortie :

- Concept_déclenché : identifiant du concept déclenché.
  format : structure ST_C_DECLENCHE.
- Code_retour : OK
  FIN_SOUS_SCHEMA : le concept déclenché est la fin d'un sous-schéma. L'identifiant du sous-schéma terminé se trouve dans Concept_déclenché.
- Code_erreur : OK

• OPERATION_CONCEPT_SUIVANT (Identifiant_op_prim, Concept_déclenché, Code_retour, Code_Erreur)

a) Spécification :

A partir de l'identifiant d'une opération primitive, cette primitive fournira le concept déclenché par cette opération primitive.

b) Entrée :

- Identifiant_op_prim : identifiant de l'opération primitive dont il faut rechercher le concept déclenché.
  format : string [MX_ID_OPERATION].

c) Sortie :

- Concept_déclenché : identifiant du concept déclenché.
  format : structure ST_C_DECLENCHE.
- Code_retour : OK
  FIN_SOUS_SCHEMA : le concept déclenché est la fin d'un sous-schéma. L'identifiant du sous-schéma terminé se trouve dans Concept_déclenché.
- Code_erreur : OK

• VARIABLE_BOUCLE (Identifiant_boucle, Variable_boucle, Code_retour, Code_erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant d'une boucle, recherche le nom de la variable associée à la boucle. La primitive est assurée que la boucle existe et qu'une variable lui est associée.

b) Entrée :

- Identifiant_boucle : identifiant de la boucle pour laquelle il faut rechercher la variable associée.
  format : string [MX_ID_BOUCLE]

c) Sortie :

- Variable_boucle : nom de la variable qui est associée à la boucle.
  format : string [MX_LG_VARIABLE]
- Code_retour : OK
- Code_erreur : OK

• CONSTITUANT_OIS (Identifiant_OIS, Type_OI, Liste_OI, Code_retour, Code_ erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant d'un objet informationnel structurant et du type des objets à sélectionner, recherche dans la base de données informationnelle tous les objets informationnels classés au premier niveau dans l'objet informationnel structurant donné et vérifiant le type demandé. Trois types de demande sont possibles :

- type = OIE : il faut sélectionner tous les objets informationnels élémentaires se trouvant dans l'O.I.S.
- type = OIS : il faut sélectionner tous les objets informationnels structurants se trouvant dans l'O.I.S.
- type = TOUS : il faut sélectionner tous les objets informationnels se trouvant dans l'O.I.S.

b) Entrée :

- Identifiant_OIS : identifiant de l'O.I.S. dont il faut retrouver les constituants.
  format : structure ST_IDENTIFIANT_OI.
- Type_OI : type des objets qu'il faut sélectionner.
  format : string [MX_LG_DESIR].

c) Sortie :

- Liste_OI : pointeur vers le premier objet informationnel sélectionné.
  format : pointeur vers une structure de type ST_LISTE_IDENTIFIANT_OI.
- Code_retour : OK
  VIDE : l'ensemble des objets informationnels sélectionné est vide
- Code_erreur : OK
  EXISTE_PAS : l'objet informationnel structurant spécifié n'existe pas

• CONSTITUANT_OIS_TOUT (Identifiant_OIS, Type_OI, Liste_OI, Code_retour, Code_ erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant d'un objet informationnel structurant et du type des objets à sélectionner, recherche dans la base de données informationnelle tous les objets informationnels classés à tous les niveaux dans l'objet informationnel structurant donné et vérifiant le type demandé. Trois types de demande sont possibles :

- type = OIE : il faut sélectionner tous les objets informationnels élémentaires se trouvant dans l'O.I.S.
- type = OIS : il faut sélectionner tous les objets informationnels structurants se trouvant dans l'O.I.S.
- type = TOUS : il faut sélectionner tous les objets informationnels se trouvant dans l'O.I.S.

b) Entrée :

- Identifiant_OIS : identifiant de l'O.I.S. dont il faut retrouver les constituants.
  format : structure ST_IDENTIFIANT_OI.
- Type_OI : type des objets qu'il faut sélectionner.
  format : string [MX_LG_DESIR].

c) Sortie :

- Liste_OI : pointeur vers le premier objet informationnel sélectionné.
  format : pointeur vers une structure de type ST_LISTE_IDENTIFIANT_OI.
- Code_retour : OK
  VIDE : l'ensemble des objets informationnels sélectionné est vide
- Code_erreur : OK
  EXISTE_PAS : l'objet informationnel structurant spécifié n'existe pas

• OBJETS_CLASSE (Classe, Liste_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive, à partir de l'identification d'une classe, recherche tous les objets informationnels qui sont de cette classe dans la base de données informationnelle.

b) Entrée :

- Classe : classe dont on veut rechercher les objets.
  format : string [MX_LG_CLASSE]

c) Sortie :

- Liste_OI : pointeur vers le premier élément sélectionné.
  format : pointeur vers une structure de type ST_LISTE_IDENTIFIANT_OI.
- Code_retour : OK
  VIDE : l'ensemble sélectionné est vide. Il n'y a pas d'objet de la classe donnée dans le système.
- Code_erreur : OK

• TOUS_OBJETS (Liste_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive recherche tous les objets informationnels du système. Il s'agit de tous les objets informationnels structurants et de tous les objets informationnels élémentaires (y compris ceux classés dans un O.I.S.).

b) Entrée :

c) Sortie :

- Liste_OI : pointeur vers le premier objet informationnel sélectionné.
  format : pointeur vers une structure de type ST_LISTE_IDENTIFIANT_OI.
- Code_retour : OK
  VIDE : l'ensemble sélectionné des objets est vide ; donc dans les système, il n'y a aucun objet informationnel.
- Code_erreur : OK

• PREMIER_OI_PILE (Nom_pile, Identifiant_oi, Code_retour, Code_erreur)

a) Spécification :

Cette primitive recherche le premier objet informationnel classé dans une pile dont on spécifie le nom et ce, en tenant compte d'un ordre de classement LIFO.

b) Entrée :

- Nom_pile : nom de la pile dont il faut rechercher le premier constituant.
  format : string [MX_LG_NOM].

c) Sortie :

- Identifiant_oi : identifiant du premier constituant de la pile.
  format : structure ST_IDENTIFIANT_OI.
- Code_retour : OK
- Code_erreur : OK
  PILE_EXISTE_PAS : la pile spécifiée n'existe pas.
  PILE_VIDE : aucun objet informationnel n'est actuellement classé dans cette pile.

• LIRE_TYPE_FICHIER (Identifiant_fichier, Classe, Type, Code_retour, Code_erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant d'un fichier va rechercher la classe et le type des objets classés dans le fichier et cela pour autant qu'un tel objet existe. Cette primitive est assurée que le fichier existe.

b) Entrée :

- Identifiant_fichier : identifiant du fichier dont il faut rechercher le type des objets classés.
  format : structure de type ST_IDENTIFIANT_OI

c) Sortie :

- Classe : classe des constituants du fichier spécifié.
  format : string [MX_LG_CLASSE].
- Type : type des constituants du fichier spécifié.
  format : string [MX_LG_TYPE]
- Code_retour : OK
  FICHIER_VIDE : le fichier spécifié est vide.
- Code_erreur : OK

• PILE_OBJET_INTERFACE (Classe, Nom, Identifiant_OIS, Code_retour, Code_erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant de la boîte IN ou de la boîte OUT va retrouver l'identifiant de la pile qui lui est associée. On est assuré que la boîte spécifiée existe.

b) Entrée :

- Classe : classe de la boîte IN ou de la boîte OUT.
  format : string [MX_OI_CLASSE].
- Nom : nom de la boîte IN ou de la boîte OUT.
  format : string [MX_OI_NOM].

c) Sortie :

- Identifiant_OIS : identifiant de la pile associée à la boîte spécifiée.
  format : structure de type ST_IDENTIFIANT_OI.
- Code_retour : OK
- Code_erreur : OK

• NUMERO_COPIE (Identifiant_OIE, Numéro, Code_retour, Code_erreur)

a) Spécification :

Cette primitive vérifie qu'il n'existe pas dans le système au moins une copie de l'O.I.E. original spécifié avec son numéro de copie supérieur algébriquement au numéro spécifié.

b) Entrée :

- Identifiant_OIE : identifiant de l'O.I.E. original.
  format : structure ST_IDENTIFIANT_OI
- Numéro : spécifie le numéro dont la valeur va être comparée aux numéros des copies de l'original.
  format : entier (2).

c) Sortie :

- Code_retour : OK
  INFERIEUR : il existe une copie dont le numéro est supérieur algébriquement à celui spécifié en paramètre.
- Code_erreur : OK

• CLASSEMENT_OI (Identifiant_OI, Identifiant_OIS, Code_retour, Code_erreur)

a) Spécification :

Cette primitive permet de tester si une objet informationnel donné est classé ou non dans un objet informationnel structurant que l'on spécifie.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel dont il faut tester s'il est classé ou non.
  format : structure ST_IDENTIFIANT_OI.
- Identifiant_OIS : identifiant de l'objet informationnel structurant.
  format : structure ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : CLASSEMENT : l'objet informationnel donné est classé dans l'O.I.S. spécifié.
  NON_CLASSEMENT : l'objet informationnel donné n'est pas classé dans l'O.I.S. spécifié.
- Code_erreur : OI_EXISTE_PAS : l'objet informationnel donné n'existe pas.
  OIS_EXISTE_PAS : l'O.I.S. spécifié n'existe pas.

• RECHERCHER_OI (Code_recherche, Classe, Type, Liste_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive à partir d'un code de recherche va retrouver tous les objets informationnels répondant au critère de recherche indiqué par le code et se trouvant en traitement sur un outil de travail. On est assuré que le code de recherche est valide.

b) Entrée :

- Code_recherche : code qui indique le critère à utiliser pour retrouver les objets informationnels dans le système. Il peut avoir les valeurs suivantes :
  1 : tous les objets informationnels se trouvant en traitement sur un outil de travail.
  2 : tous les objets informationnels d'une classe donnée se trouvant en traitement sur un outil de travail.
  3 : tous les objets informationnels d'une classe donnée et d'un type donné se trouvant en traitement sur un outil de travail.
  format : entier (1).
- Classe : classe des objets informationnels qu'il faut rechercher. Ce paramètre n'a de sens que si le code de recherche appartient à (2, 3).
  format : string [MX_LG_CLASSE].
- Type : type des objets informationnels qu'il faut rechercher. Ce paramètre n'a de sens que si le code de recherche appartient à (3).
  format : string [MX_LG_TYPE].

c) Sortie :

- liste_OI : ensemble des objets informationnels répondant au critère indiqué par le code de recherche.
  format : pointeur vers une structure de type ST_LISTE_IDENTIFIANT_OI.
- Code_retour : OK
- Code_erreur : OK

• TRAVAIL_OI (Identifiant_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive doit regarder si l'objet informationnel spécifé se trouve ou non sur un outil de travail. On est assuré que l'objet informationnel existe.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel.
  format : structure de type ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
  TABLE_DE_TRAVAIL : l'objet informationnel se trouve sur la table de travail.
  PHOTOCOPIEUSE : l'objet informationnel se trouve sur la photocopieuse.
- Code_erreur : OK

• CREER_OI (Valeur_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de créer un objet informationnel dans le système. L'association COPIE est gérée par cette primitive ; si l'objet créé est un original alors il sera associé à ses copies éventuelles existantes et si l'objet créé est une copie alors elle sera associée à son original s'il existe.

b) Entrée :

- Valeur_OI : ensemble des informations directement relatives à l'objet informationnel que l'on doit créer.
  format : structure de type ST_VALEUR_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• MODIFIER_OI (Valeur_OI, B_état, B_échéance, B_contenu, B_ordre, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour but de modifier les informations qui se trouvent mémorisées dans le système à propos d'un objet informationnel. Les informations suivantes ne seront cependant pas modifiées : la classe, le type , le nom et le numéro de copie.

L'ordre de classement, l'état et l'échéance seront modifiés uniquement si les paramètres le signalent.

b) Entrée :

- Valeur_OI : identifiant de l'objet informationnel à modifier et nouvelles valeurs des champs à modifier.
  format : structure de type ST_VALEUR_OI.
- B_état : indique si l'état doit être mis à jour.
  format : entier (1). Ensemble des valeurs : {VRAI, FAUX}
- B_échéance : indique si l'échéance doit être mise à jour.
  format : entier (1). Ensemble des valeurs : {VRAI, FAUX}
- B_contenu : indique si l'état contenu doit être mis à jour.
  format : entier (1). Ensemble des valeurs : {VRAI, FAUX}
- B_ordre : indique si l'ordre de classement doit être mis à jour.
  format : entier (1). Ensemble des valeurs : {VRAI, FAUX}

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• LECTURE_OI (Identifiant_OI, Info_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant d'un objet informationnel, vérifie que celui-ci existe dans le système. Dans le cas où il existe, la primitive fournit les informations le concernant.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel dont on veut tester l'existence dans le système et lire les informations.
  format : struture ST_IDENTIFIANT_OI

c) Sortie :

- Info_OI : ensemble des informations relatives à l'objet informationnel.
  format : structure de type ST_VALEUR_OI.
- Code_retour : OK
  EXISTE_PAS : l'objet informationnel spécifié n'existe pas.
- Code_erreur : OK

• LECTURE_OBJET (Classe_objet, Nom_objet, Type_objet, Code_retour, Code_erreur)

a) Spécification :

Cette primitive, à partir de l'identifiant d'un objet de rangement ou d'un objet d'interface ou encore d'un outil de travail, vérifie que cet objet existe dans le système.

b) Entrée :

- Classe_objet : classe de l'objet ou de l'outil dont on veut vérifier l'existence dans le système.
  format : string [MX_OBJET_CLASSE]
- Nom_objet : nom de l'objet ou de l'outil dont on veut tester l'existence dans le système.
  format : string [MX_OBJET_NOM]
- Type_objet : indique si ce que l'on recherche est un objet de rangement, un objet d'interface ou un outil de travail.
  format : entier (1). Ensemble des valeurs :
  {OBJET_RANGEMENT, OBJET_INTERFACE, OUTIL_TRAVAIL}.

c) Sortie :

- Code_retour : OK
  EXISTE_PAS : l'objet ou l'outil spécifié n'existe pas.

- Code_erreur : OK

• OUTIL_TRAVAIL_OI (Identifiant_OI, Classe, Code_retour, Code_erreur)

a) Spécification :

Cette primitive va associer un objet informationnel à un outil de travail. On est assuré que l'objet informationnel existe, de même que l'outil de travail. On est aussi assuré que l'objet informationnel ne se trouve pas déjà sur un outil de travail.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à associer à l'outil de travail.
  format : structure de type ST_IDENTIFIANT_OI.
- Classe : identifiant de l'outil de travail auquel il faut associer un objet informationnel.
  format : string [MX_OT_CLASSE].

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• INTERFACE_OI (Identifiant_OI, Classe, Nom, Code_retour, Code_erreur)

a) Spécification :

Cette primitive va associer un objet informationnel à un objet d'interface. On est assuré que l'objet informationnel existe, de même que l'objet d'interface. On est aussi assuré que l'objet informationnel n'est pas déjà associé à un objet d'interface.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à associer à un objet d'interface.
  format : structure de type ST_IDENTIFIANT_OI.
- Classe : classe de l'objet d'interface auquel il faut associer à un objet informationnel.
  format : string [MX_OI_CLASSE].
- Nom : nom de l'objet d'interface auquel il faut associer à un objet informationnel.
  format : string [MX_OI_NOM].

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• CLASSER_OI (Identifiant_OI, Identifiant_OIS, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de classer un objet informationnel spécifié dans un objet informationnel structurant. On est assuré que l'objet informationnel et que l'objet informationnel structurant existent.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à classer.
  format : structure de type ST_IDENTIFIANT_OI.
- Identifiant_OIS : identifiant de l'objet informationnel structurant.
  format : structure de type ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• RANGER_OI (Identifiant_OI, Classe, Nom, Code_retour, Code_erreur)

a) Spécification :

Cette primitive a pour objectif d'associer l'objet informationnel spécifié avec l'objet de rangement associé. On est assuré que l'objet informationnel existe ainsi que l'objet de rangement.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à ranger.
  format : structure de type ST_IDENTIFIANT_OI.
- Classe : classe de l'objet de rangement.
  format : string [MX_OR_CLASSE].
- Nom : nom de l'objet de rangement.
  format : string [MX_OR_NOM].

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• REPLACER (Identifiant_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive va replacer l'objet informationnel spécifié dans son objet de rangement, dans son objet d'interface ou encore, dans son objet de classement. On est assuré que l'objet informationnel existe et qu'il se trouve sur un outil de travail.

Si l'objet spécifié se trouve sur un outil de travail mais qu'il ne possède pas d'endroit où il puisse être replacé, alors pour cet objet, la primitive ne va pas le replacer.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à replacer.
  format : structure de type ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• REPLACER_OI (Identifiant_OI, Code_retour, Code_erreur)

a) Spécification :

Cette primitive va replacer l'objet informationnel spécifié dans son objet de rangement, dans son objet interface ou encore, dans son objet de classement. On est assuré que l'objet informationnel existe et qu'il se trouve sur un outil de travail.

Cette primitive supprime l'association TRAITEMENT dans tous les cas ; même si l'objet ne se trouve associé à aucun autre objet dans le bureau. C'est au module de niveau supérieur d'agir en conséquence pour respecter la contrainte d'intégrité suivante : tout objet informationnel doit participer à au moins une association autre que l'association COPIE.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à replacer.
  format : structure de type ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• REPLACER_CONST (Identifiant_OIS, Code_retour, Code_erreur)

a) Spécification :

Cette primitive à partir de l'identifiant d'un O.I.S., va s'assurer que tous les constituants de cet O.I.S. se trouvent physiquement dans celui-ci. Cela veut dire que lorsque l'exécution est terminée, plus aucun constituant de l'O.I.S. ne se trouve en traitement sur un outil de travail. On est assuré que l'O.I.S. spécifié existe.

b) Entrée :

- Identifiant_OIS : identifiant de l'objet informationnel structurant.
  format : structure de type ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

• DETACHER_OI (Identifiant_OI, Code_retour, Code_erreur)

a) Spécification

Cette primitive à partir de l'identifiant d'un objet informationnel doit assurer qu'après son exécution :

- l'objet informationnel spécifié ne sera plus en traitement sur un outil de travail,
- l'objet informationnel spécifié ne sera plus rangé dans un objet de rangement,
- l'objet informationnel spécifié ne sera plus dans un objet d'interface,
- l'objet informationnel spécifié ne sera plus classé dans un autre objet informationnel structurant.

Par contre, l'objet informationnel pourra encore être associé soit à un original, soit à ses copies. On est assuré que l'objet informationnel existe.

N.B. : Si l'objet informationnel est un O.I.S., alors ses constituants ne seront pas traités. Il se peut donc que ceux-ci restent en traitement sur un outil de travail.

b) Entrée :

- Identifiant_OI : identifiant de l'objet informationnel à detacher.
  format : structure de type ST_IDENTIFIANT_OI.

c) Sortie :

- Code_retour : OK
- Code_erreur : OK

3.19 Module SORTIE_ECRAN

• SORTIE_ECRAN_CREER (Information_créer, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive créer. L'O.I. à créer doit être amené sur la table de travail.

b) Entrée :

- Information_créer : donne les informations utiles à l'opération créer.
  format : structure ST_SE_CREER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_REPRODUIRE (Information_reproduire, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive reproduire. L'original est placé sur la photocopieuse s'il ne s'y trouve déjà ainsi que toutes les copies.

b) Entrée :

- Information_reproduire : donne les informations utiles à l'opération reproduire.
  format : structure ST_SE_REPRODUIRE.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_RECEVOIR (Information_recevoir, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive recevoir. S'il s'agit d'une réception écrite, l'objet reçu est alors placé dans la boîte IN. S'il s'agit d'une réception écrite avec trace écrite, le message est placé à côté du téléphone.

b) Entrée :

- Information_recevoir : donne les informations utiles à l'opération recevoir.
  format : structure ST_SE_RECEVOIR.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_COMMUNIQUER (Information_communiquer, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive communiquer. L'objet à communiquer est placé dans la boîte OUT s'il s'agit d'une communication écrite sinon une communication téléphonique est simulée.

b) Entrée :

- Information_communiquer : donne les informations utiles à l'opération communiquer.
  format : structure ST_SE_COMMUNIQUER..

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_CONSULTER (Information_consulter, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive consulter. L'O.I. à consulter est amené sur la table de travail s'il ne s'y trouve pas déjà.

b) Entrée :

- Information_consulter : donne les informations utiles à l'opération consulter.
  format : structure ST_SE_CONSULTER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_FEUILLETER (Information_feuilleter, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive feuilleter. L'O.I.S. à feuilleter est amené sur la table de travail et tous ses constituants sont affichés.

b) Entrée :

- Information_feuilleter : donne les informations utiles à l'opération feuilleter.
  format : structure ST_SE_FEUILLETER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_ENLEVER (Information_enlever, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive enlever. L'O.I. à enlever de la pile est placé sur la table de travail.

b) Entrée :

- Information_enlever : donne les informations utiles à l'opération enlever.
  format : structure ST_SE_ENLEVER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_MODIFIER (Information_modifier, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive modifier. L'O.I. à modifier est placé sur la table de travail s'il ne s'y trouve déjà ainsi que le message si celui-ci est spécifié et qu'il ne s'y trouve pas.

b) Entrée :

- Information_modifier : donne les informations utiles à l'opération modifier.
  format : structure ST_SE_MODIFIER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_COMPLETER (Information_compléter, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive compléter. Le formulaire à compléter est placé sur la table de travail s'il ne s'y trouve déjà.

b) Entrée :

- Information_compléter : donne les informations utiles à l'opération compléter.
  format : structure ST_SE_COMPLETER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_VERIFIER (Information_vérifier, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive vérifier. L'O.I. à vérifier est placé sur la table de travail s'il ne s'y trouve déjà.

b) Entrée :

- Information_vérifier : donne les informations utiles à l'opération vérifier.
  format : structure ST_SE_VERIFIER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_DETRUIRE (Information_détruire, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive détruire. L'O.I. à détruire est placé dans la poubelle.

b) Entrée :

- Information_détruire : donne les informations utiles à l'opération détruire.
  format : structure ST_SE_DETRUIRE.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_ARCHIVER (Information_archiver, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive archiver. L'O.I. à archiver est placé dans la boîte à archives.

b) Entrée :

- Information_archiver : donne les informations utiles à l'opération archiver.
  format : structure ST_SE_ARCHIVER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_TRIER (Information_trier, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive trier. L'O.I.S. à trier est amené sur la table de travail s'il ne s'y trouve déjà.

b) Entrée :

- Information_trier : donne les informations utiles à l'opération trier.
  format : structure ST_SE_TRIER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_CLASSER (Information_classer, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive classer. L'O.I.S. à classer est amené dans l'objet de classement.

b) Entrée :

- Information_classer : donne les informations utiles à l'opération classer.
  format : structure ST_SE_CLASSER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_RANGER (Information_ranger, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive ranger. L'O.I. à ranger est amené dans son objet de rangement.

b) Entrée :

- Information_ranger : donne les informations utiles à l'opération ranger.
  format : structure ST_SE_RANGER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_REPLACER (Information_replacer, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive replacer. Les O.I. à replacer sont dirigés vers leur objet de rangement, d'interface ou de classement.

b) Entrée :

- Information_replacer : donne les informations utiles à l'opération replacer.
  format : structure ST_SE_REPLACER.

c) Sortie :

- Code_erreur : OK

• SORTIE_ECRAN_DECIDER (Information_décider, Code_erreur)

a) Spécification :

Cette primitive a pour objectif de simuler graphiquement une opération primitive décider. L'O.I. à partir duquel se fait l'opération est amené sur la table de travail s'il ne s'y trouve pas déjà.

b) Entrée :

- Information_décider : donne les informations utiles à l'opération décider.
  format : structure ST_SE_DECIDER.

c) Sortie :

- Code_erreur : OK

Annexe 4
Transformation du schéma entité-association en schéma conforme

ANNEXE 4 : TRANSFORMATION DU SCHEMA ENTITE-ASSOCIATION EN SCHEMA CONFORME

4.1 Transformation des schémas entité-association en schémas des accès nécessaires

L'implémentation du système exige de transformer les schémas conceptuels du bureau et du langage dans un schéma respectant les possibilités d'un Système de Gestion de Données (S.G.D.).

Pour y arriver, plusieurs étapes seront suivies :

- traduction du schéma conceptuel dans un schéma des accès possibles,
- traduction du schéma des accès posssibles en schéma des accès nécessaires.
- traduction du schéma des accès nécessaires en schéma conformes au S.G.D. utilisé.

Les règles de transformation pour passer d'un schéma à l'autre sont décrites dans [HAIN 86].

A partir des différentes fonctions d'accès au système d'information décrites dans l'annexe 3, nous reporterons sur nos schémas des accès possibles, les accès qui seront utilisés par ces fonctions. Nous obtenons alors les schémas des accès nécessaires donnés ci-dessous.

4.1.1 Transformation du schéma conceptuel du bureau

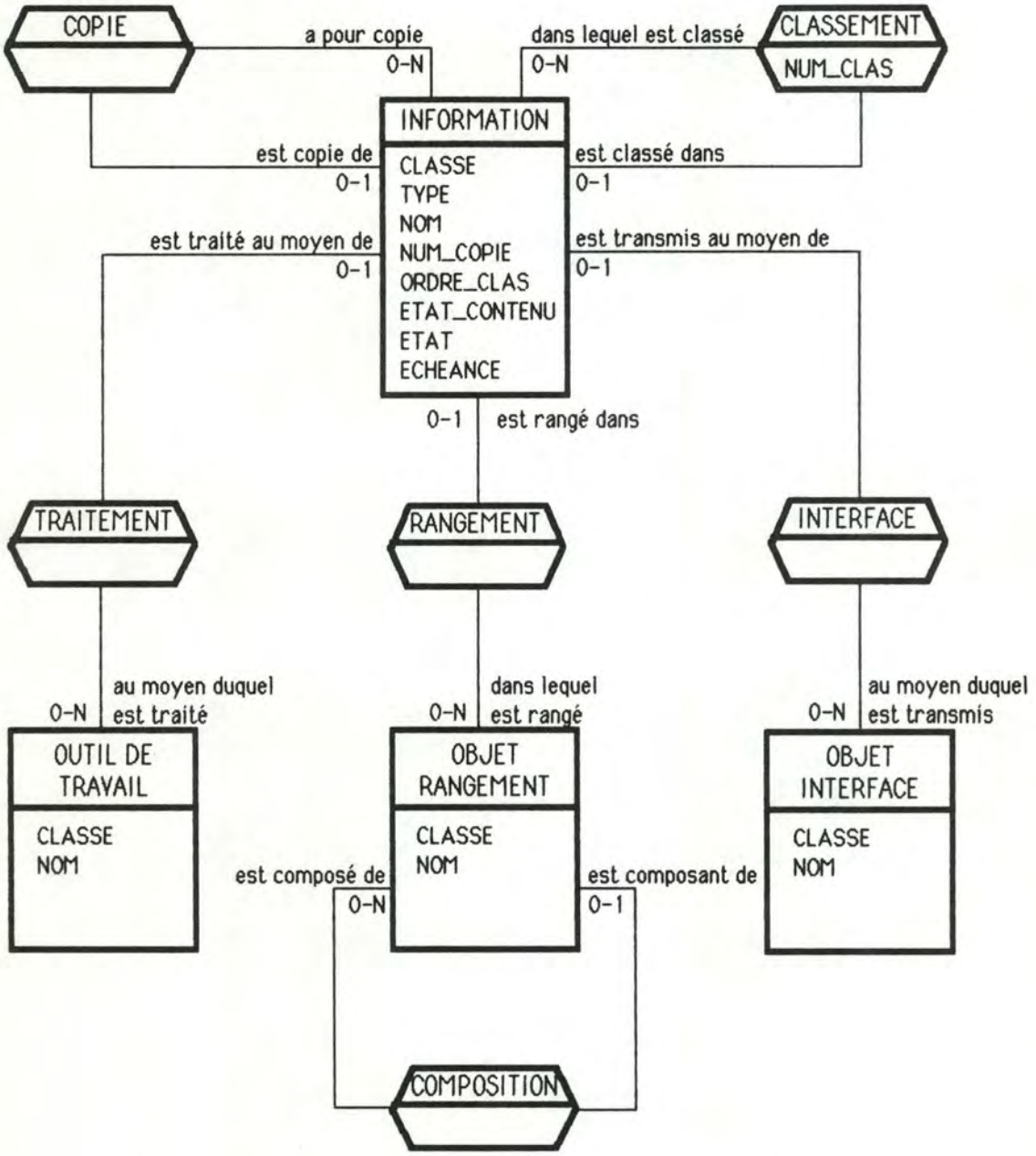


Fig A4.1 : Schéma conceptuel du bureau (décrit au point 3.8)

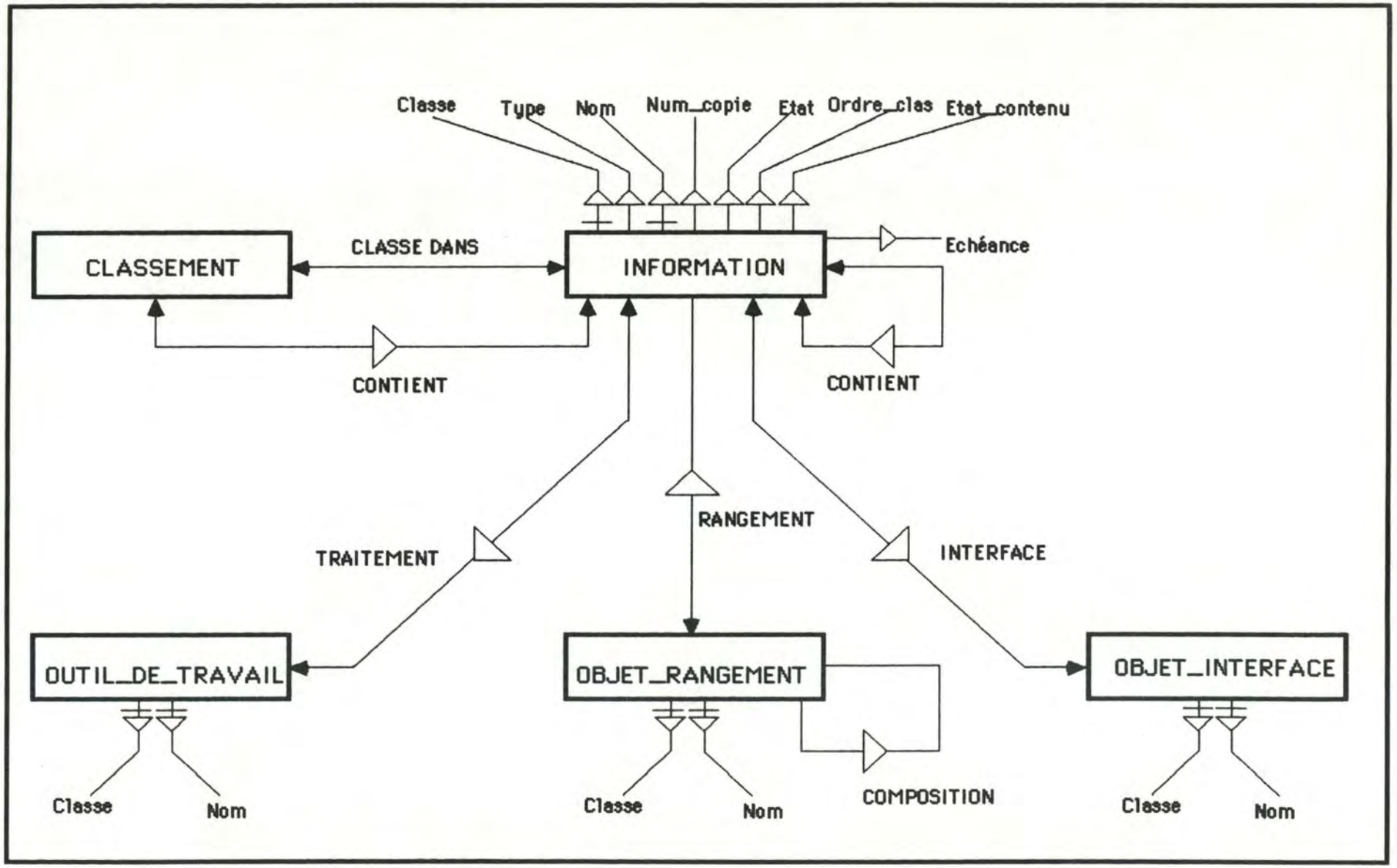


Fig A4.2 : Schéma des accès nécessaires pour la base de données du bureau

4.1.2 Transformation du schéma conceptuel du langage

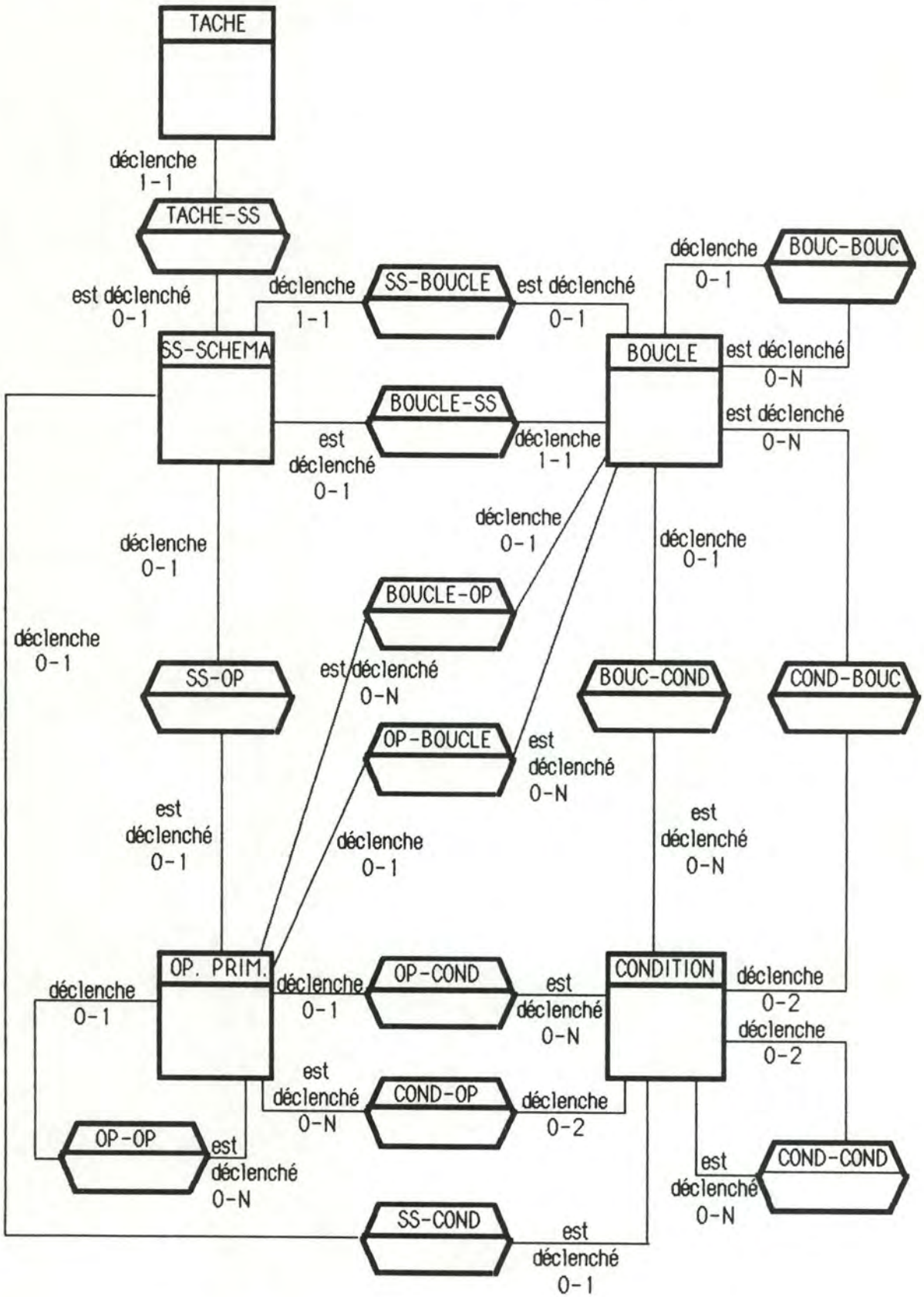


Fig A4.3 : Schéma conceptuel du langage (décrit au point 5.2)

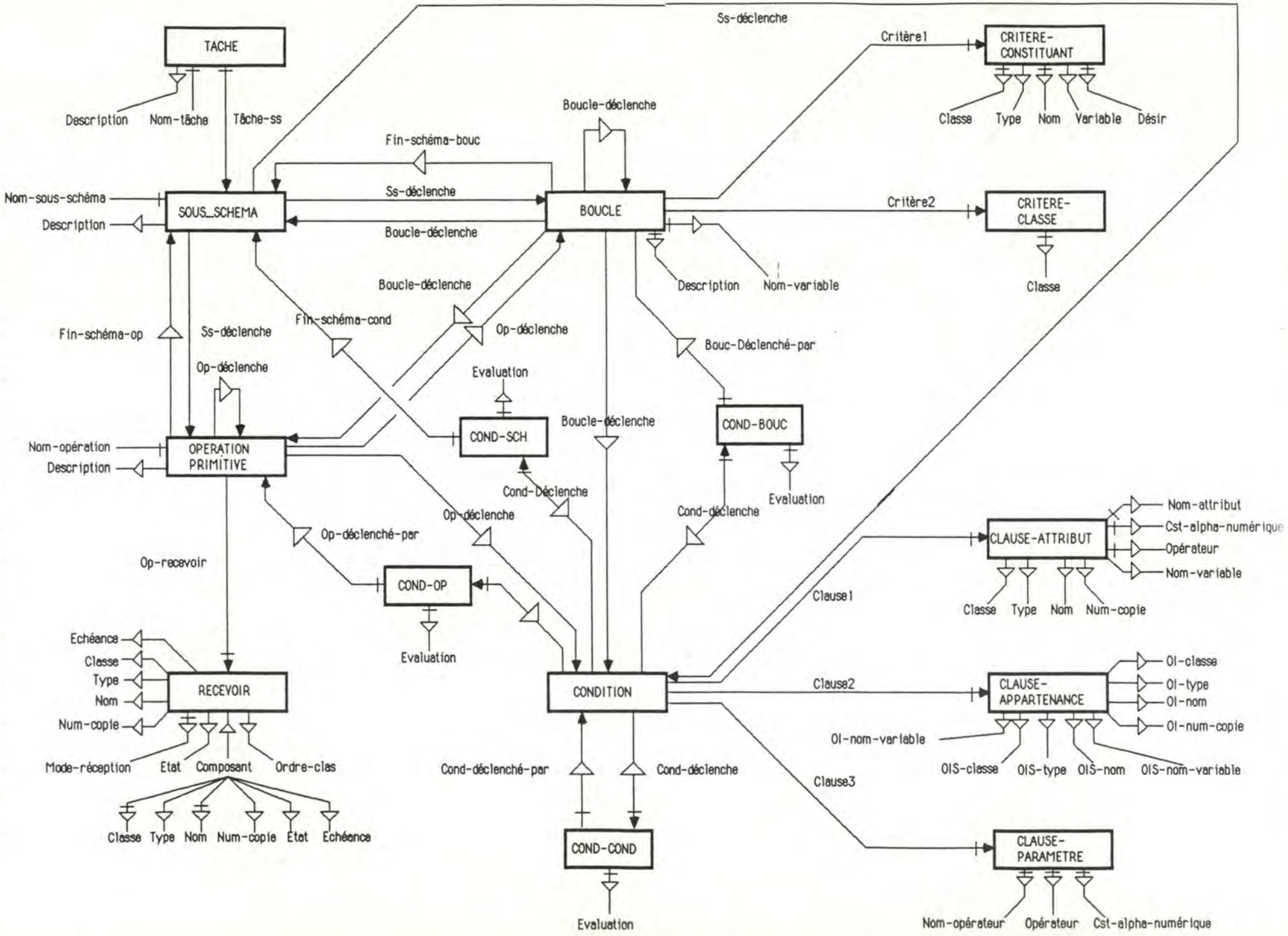


Fig A4.4 : Schéma des accès nécessaires pour la base de données du langage

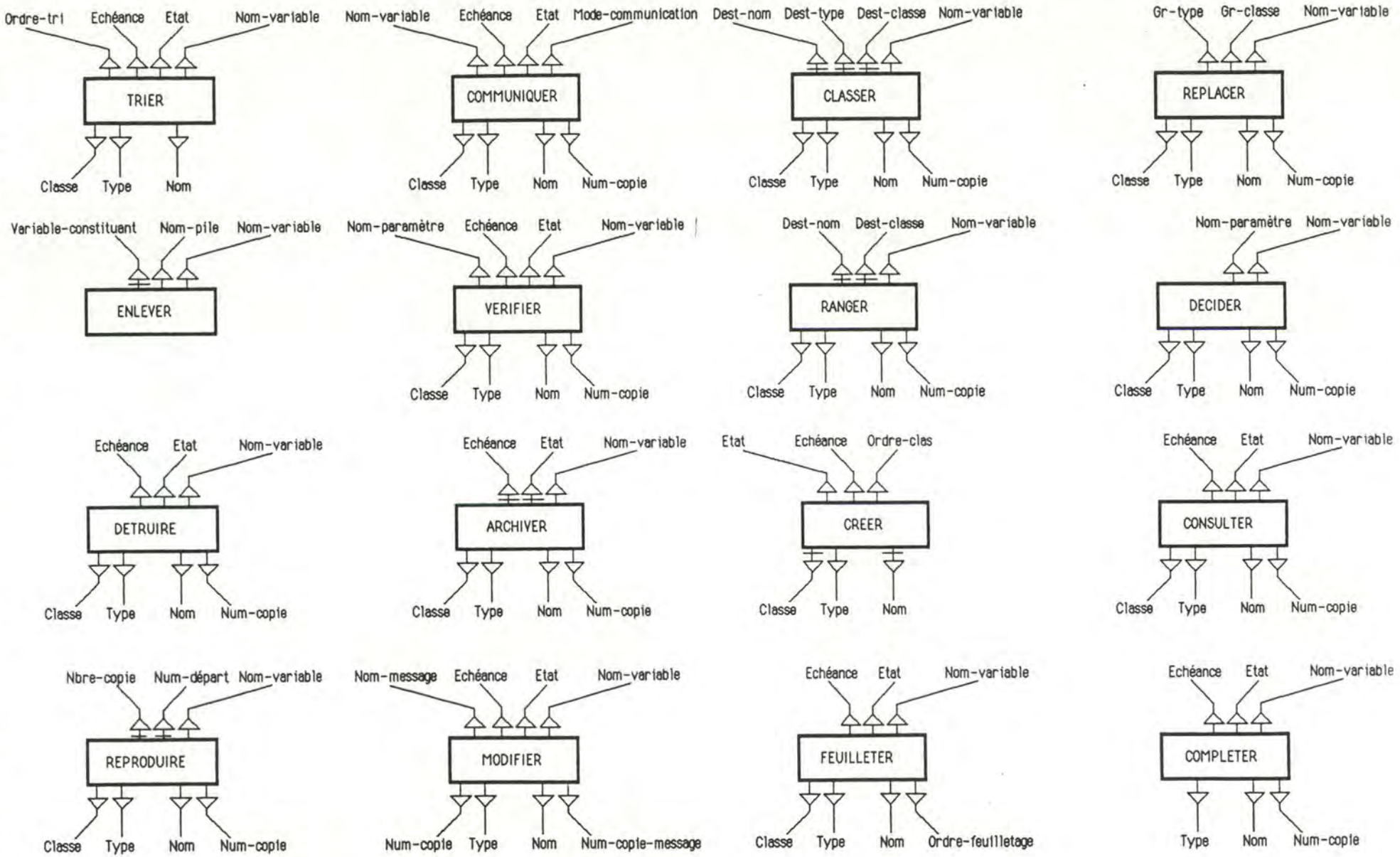


Fig.A4.4 : Schéma des accès nécessaires pour la base de données du langage

4.2 Transformation des schémas des accès nécessaires en schémas conformes

Le choix du SGD (Système de Gestion des Données) a dû se faire en tenant compte du gestionnaire graphique utilisé et de la portabilité de notre système.

Pour ces raisons, notre choix c'est porté sur les fichiers séquentiels en C. D'une part, les primitives utilisées sont les primitives standards du C et donc utilisables avec tout compilateur C et d'autre part, pour le moment, certains logiciels graphiques ne sont pas compatibles avec d'autres SGD.

Les caractéristiques des fichiers séquentiels C :
- pas de chemins d'accès aux données,
- pas de clés d'accès,
- pas d'items répetitifs,
- utilisation possible d'items décomposables.

Les accès possibles :
- lecture séquentielle,
- écriture en extension ou non,
- modification d'un article,
- pas de suppression,
- positionnement sur un article spécifié.

Les modes d'ouverture des fichiers :
- lecture seule,
- écriture seule,
- lecture et écriture à partir d'un nouveau fichier (il existe ou non). Si le fichier existe alors son contenu est détruit,
- lecture et écriture en extension.

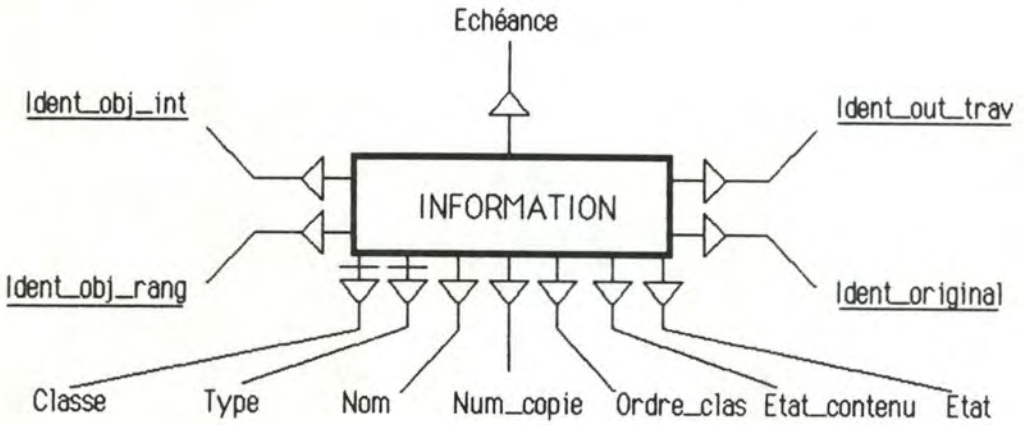
Conventions d'écriture

Les items soulignés **0** fois sont les items initiaux tel qu'ils étaient dans le schéma conceptuel.

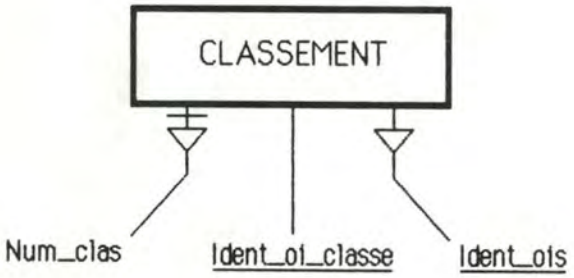
Les items soulignés **1** fois sont ceux obtenus par rotation (la rotation est une des règles de transformation décrite dans [HAIN 86]).

Les items soulignés **2** fois sont ceux que nous avons ajoutés pour des raisons de performance. Par exemple, l'item TYPE_CONCEPT_DECLENCHE dans le type d'article OPERATION_PRIMITIVE, indique quel chemin d'accès il faut utiliser pour retrouver le concept déclenché par une opération primitive. Un autre exemple est l'item TYPE_CRITERE dans l'article BOUCLE qui indique quel est le type de critère de la boucle.

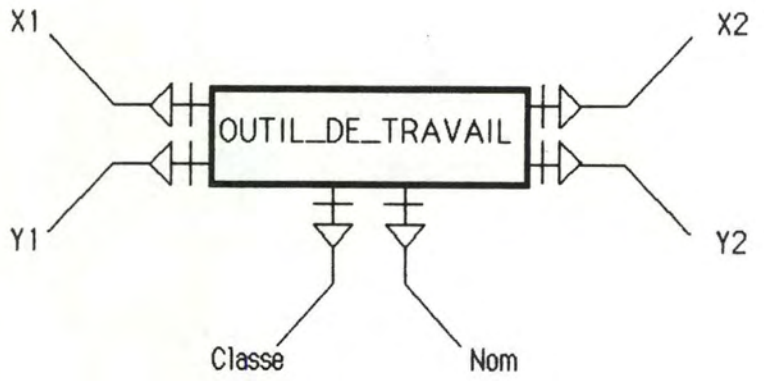
4.2.1 Schéma conforme pour la base de données du bureau

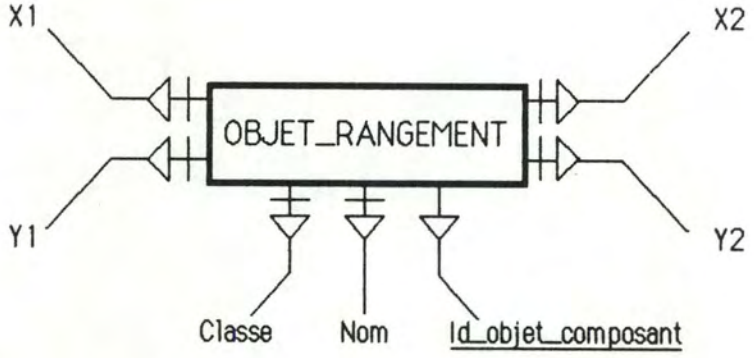


[Information.ident_obj_int] in [Objet_interface.classe + nom]
[Information.ident_out_trav] in [Outil_de_travail.classe + nom]
[Information.ident_obj_rang] in [Objet_rangement.classe + nom]
[Information.ident_original] in [Information.classe + type + nom +num_copie]

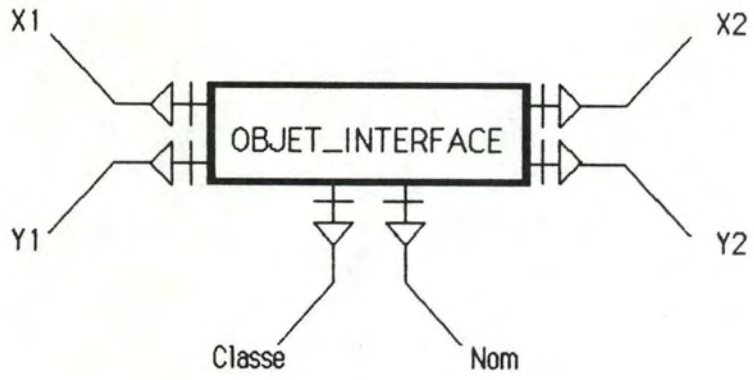


[Classement.ident_oi_classe] in [Information.classe + type + nom + num_copie]
[Classement.ident_ois] in [Information.classe + type + nom + num_copie]



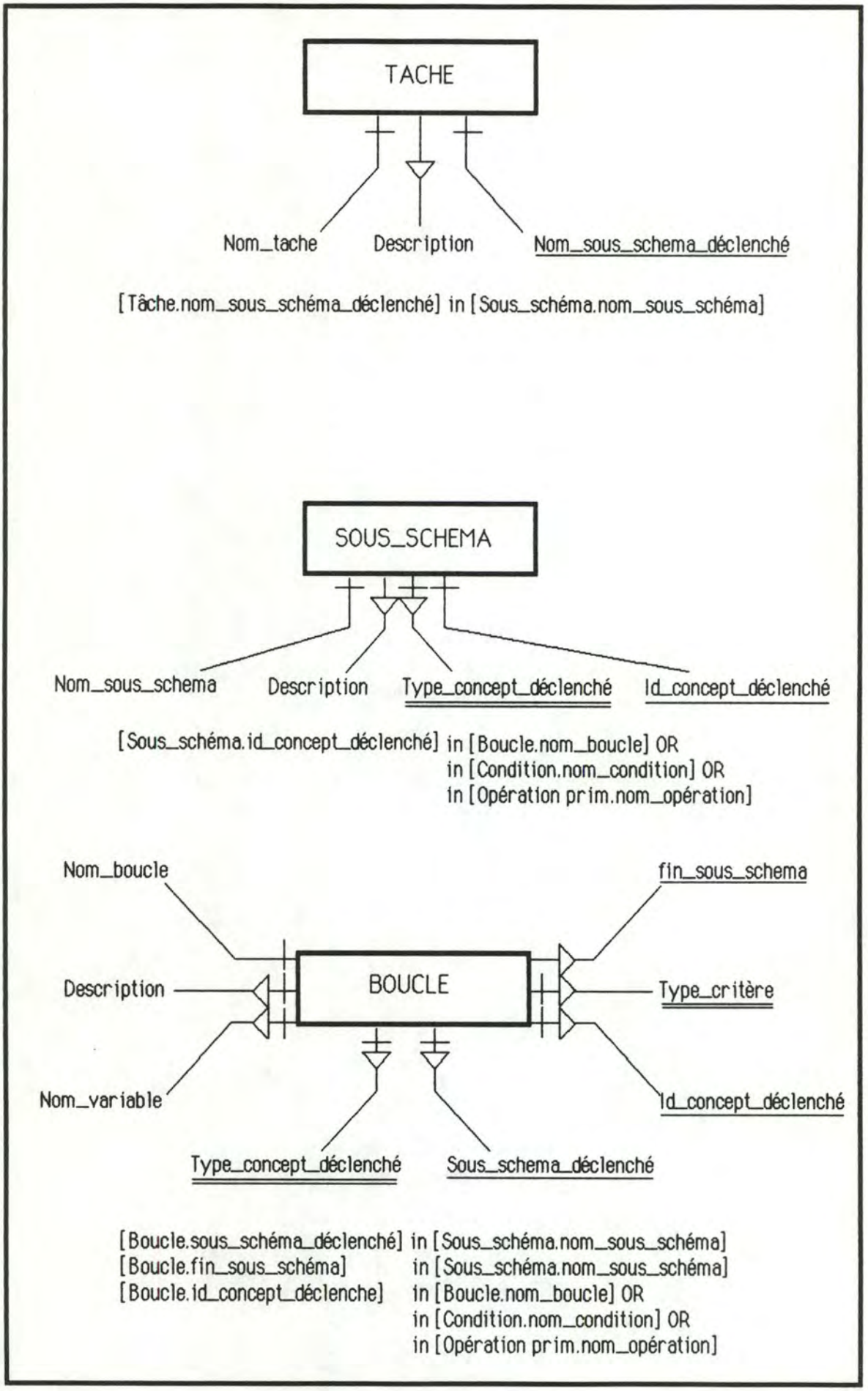


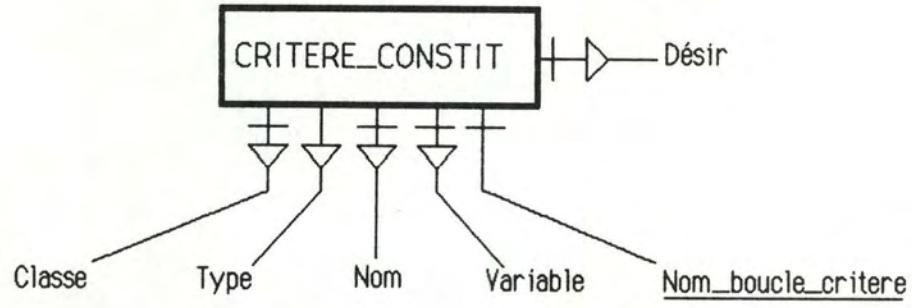
[Objet_rangement.id_objet_composant] in [Objet_rangement.classe + nom]



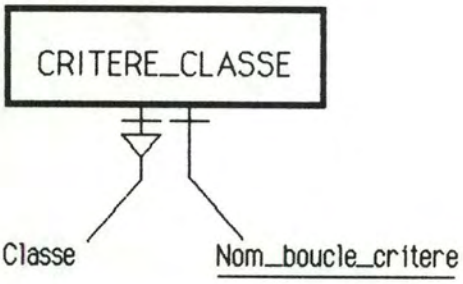
4.2.2 Schéma conforme pour la base de données du langage

En ce qui concerne la transformation des articles concernant les opérations, nous ne donnerons le schéma que de l'opération RECEVOIR car d'une part c'est la plus compliquée des opérations à transformer du fait d'un item répétitif et ensuite toutes les autres transformations d'opérations suivent le même principe. Il est donc inutile et sans intérêts de les décrire toutes.

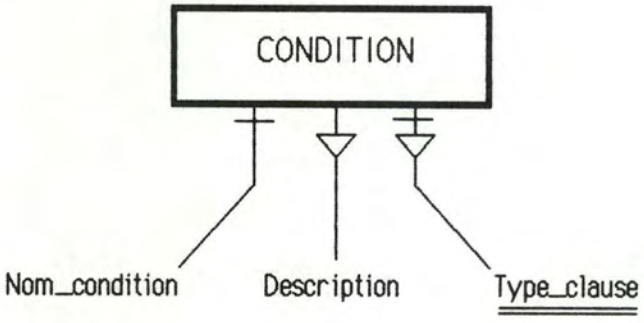

TACHE
Nom_tache
Description
Nom_sous_schema_déclenché
[Tâche.nom_sous_schéma_déclenché] in [Sous_schéma.nom_sous_schéma]
SOUS_SCHEMA
Nom_sous_schema
Description
Type_concept_déclenché
Id_concept_déclenché
[Sous_schéma.id_concept_déclenché] in [Boucle.nom_boucle] OR
in [Condition.nom_condition] OR
in [Opération prim.nom_opération]
Nom_boucle
fin_sous_schema
Description
BOUCLE
Type_critère
Nom_variable
Id_concept_déclenché
Type_concept_déclenché
Sous_schema_déclenché
[Boucle.sous_schéma_déclenché] in [Sous_schéma.nom_sous_schéma]
[Boucle.fin_sous_schéma] in [Sous_schéma.nom_sous_schéma]
[Boucle.id_concept_déclenche] in [Boucle.nom_boucle] OR
in [Condition.nom_condition] OR
in [Opération prim.nom_opération]

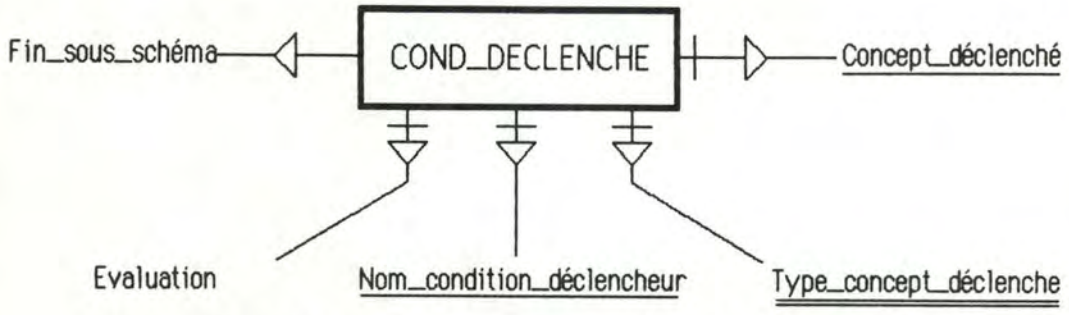


[Critère_constit.nom_boucle_critère] in [Boucle.nom_boucle]

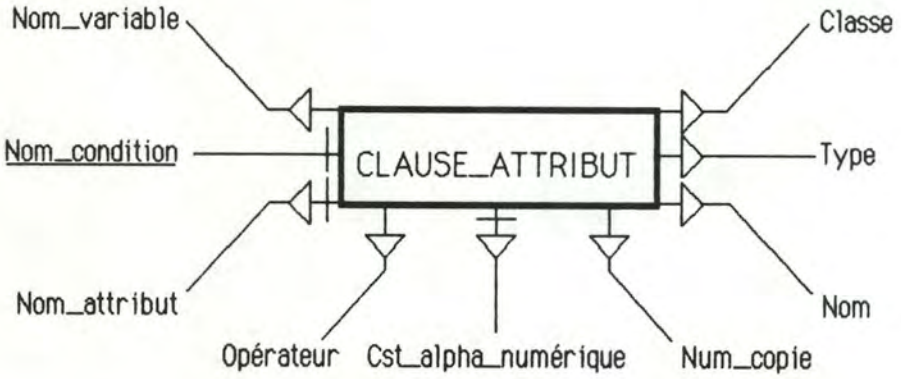


[Critère_classe.nom_boucle_critère] in [Boucle.nom_boucle]

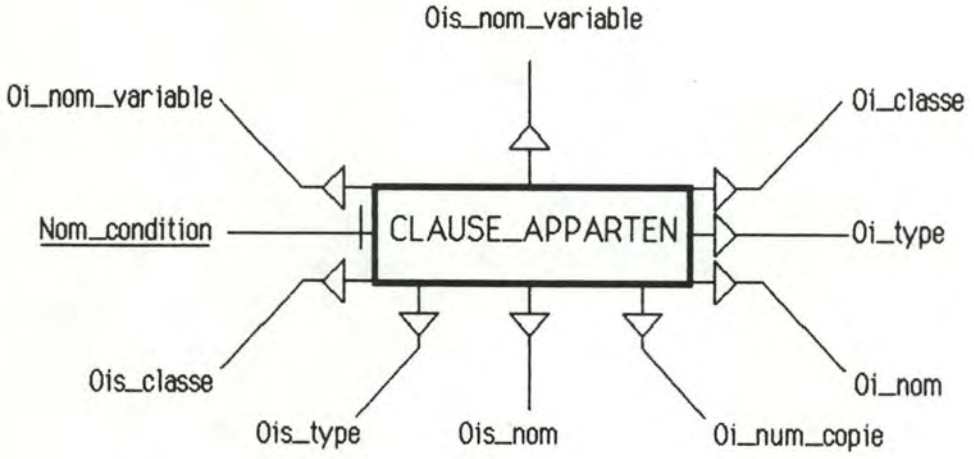




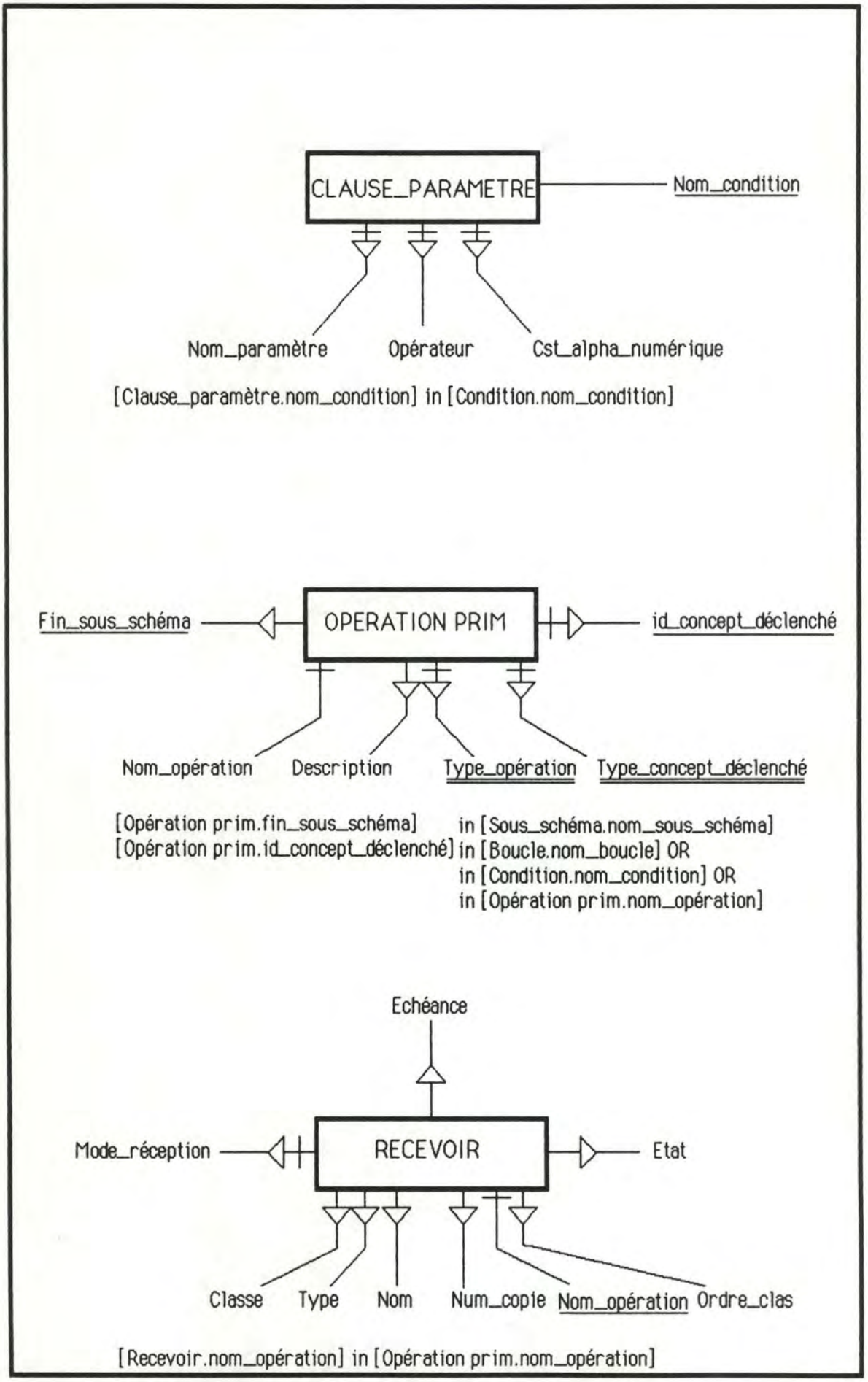
[Cond_déclenche.nom_condition_déclencheur] in [Condition.nom_condition]
[Cond_déclenche.concept_declenche] in [Boucle.nom_boucle] OR
in [Condition.nom_condition] OR
in [Opération prim.nom_opération]

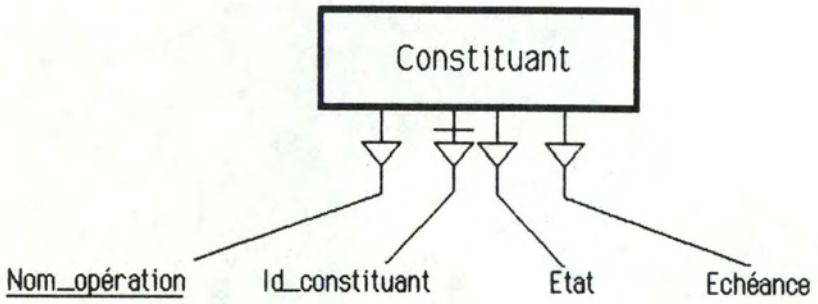


[Clause_attribut.nom_condition] in [Condition.nom_condition]



[Clause_apparten.nom_condition] in [Condition.nom_condition]


CLAUSE_PARAMETRE
Nom_condition
Nom_paramètre
Opérateur
Cst_alpha_numérique
[Clause_paramètre.nom_condition] in [Condition.nom_condition]
Fin_sous_schéma
OPERATION PRIM
id_concept_déclenché
Nom_opération
Description
Type_opération
Type_concept_déclenché
[Opération prim.fin_sous_schéma] in [Sous_schéma.nom_sous_schéma]
[Opération prim.id_concept_déclenché] in [Boucle.nom_boucle] OR
in [Condition.nom_condition] OR
in [Opération prim.nom_opération]
Echéance
Mode_réception
RECEVOIR
Etat
Classe
Type
Nom
Num_copie
Nom_opération
Ordre_clas
[Recevoir.nom_opération] in [Opération prim.nom_opération]



[Constituant.nom_opération] in [Recevoir.nom_opération]

4.3 Schéma interne

A partir du schéma conforme de chaque base de données, nous en donnons maintenant le schéma interne. Nous y décrivons la répartition des types d'articles dans les fichiers et la description de chaque fichier obtenu.

4.3.1 Schéma interne de la base de données du bureau

Chaque type d'article se trouve dans un fichier texte en C.

• Structure du fichier INFORMATION : ST_BD_INFORMATION

- Identifiant_oi = structure ST_IDENTIFIANT_OI.
- Etat = string [MX_LG_ETAT].
- Echéance = string [MX_LG_ECHEANCE].
- Ordre_clas = string [MX_LG_ORDRE].
- Etat_contenu = string [MX_LG_ETAT_CONTENU].
- Identifiant_original = structure ST_IDENTIFIANT_OI.
- Obj_int_classe = string [MX_OI_CLASSE].
- Obj_int_nom = string [MX_OI_NOM].
- Obj_rang_classe = string [MX_OR_CLASSE].
- Obj_rang_nom = string [MX_OR_NOM].
- Obj_trav_classe = string [MX_OT_CLASSE].
- Obj_trav_nom = string [MX_OT_NOM].

• Structure du fichier CLASSEMENT : ST_BD_CLASSEMENT

- Num_clas = entier (2).
- Id_oi_classe = structure ST_IDENTIFIANT_OI.
- Id_ois = structure ST_IDENTIFIANT_OI.

• Structure du fichier OUTIL_TRAV : ST_BD_OUTIL_TRAV

- Classe = string [MX_OT_CLASSE].
- Nom = string [MX_OT_NOM].
- X1 = entier (2).
- Y1 = entier (2).
- X2 = entier (2).
- Y2 = entier (2).

• Structure du fichier OBJ_RANGEMENT : ST_BD_OBJ_RANGEMENT

- Classe = string [MX_OR_CLASSE].
- Nom = string [MX_OR_NOM].
- Comp_classe = string [MX_OR_CLASSE].

- Comp_nom = string [MX_OR_NOM].
- X1 = entier (2).
- Y1 = entier (2).
- X2 = entier (2).
- Y2 = entier (2).

• Structure du fichier OBJ_INTERFACE : ST_BD_OBJ_INTERFACE

- Classe = string [MX_OI_CLASSE].
- Nom = string [MX_OI_NOM].
- X1 = entier (2).
- Y1 = entier (2).
- X2 = entier (2).
- Y2 = entier (2).

4.3.2 Schéma interne de la base de données du langage

Nous avons regroupé plusieurs types d'articles dans un même fichier en vue de limiter le nombre de fichier. Les types d'article regroupés dans un même fichier ayant un caractère sémantique identique.

Nous avons regroupé :

- les types d'articles CRITERE_CONSTIT et CRITERE_CLASSE dans le fichier CRITERE.
- les types d'articles CLAUSE_ATTRIBUT, CLAUSE_APPARTEN et CLAUSE_PARAMETRE dans le fichier CLAUSE.
- le type d'article CONSTITUANT et tous les types d'article représentant une opération primitive dans le fichier OPERATION.

• Structure du fichier TACHE : ST_BD_TACHE

- Nom_tache = string [MX_ID_TACHE].
- Description = string [MX_LG_DESCRIPTION].
- Nom_sous_schéma_déclenché = string [MX_ID_SOUS_SCHEMA].

• Structure du fichier SOUS_SCHEMA : ST_BD_SOUS_SCHEMA

- Nom_sous_schéma = string [MX_ID_SOUS_SCHEMA].
- Description = string [MX_LG_DESCRIPTION].
- Type_concept_déclenché = entier (2).
- Id_concept_déclenché = string [MX_ID_CONCEPT].

• Structure du fichier BOUCLE : ST_BD_BOUCLE

- Nom_boucle = string [MX_ID_BOUCLE].

- Description = string [MX_LG_DESCRIPTION].
- Nom_variable = string [MX_LG_VARIABLE].
- Fin_sous_schéma = string [MX_ID_SOUS_SCHEMA].
- Type_critère = entier (2).
- Type_concept_déclenché = entier (2).
- Id_concept_déclenché = string [MX_ID_CONCEPT].
- Sous_schéma_déclenché = string [MX_ID_SOUS_SCHEMA].

● Structure du fichier CRITERE : ST_BD_CRITERE

- Nom_boucle_critère = string [MX_ID_BOUCLE].
- Type_critère : champ variable composé de :

  ° Constituant : structure composée de :

  - Classe = string [MX_LG_CLASSE].
  - Type = string [MX_LG_TYPE].
  - Nom = string [MX_LG_NOM].
  - Variable = string [MX_LG_VARIABLE].
  - Désir = string [MX_LG_DESIR].

  ° Classe : structure composée de :

  - Classe = string [MX_LG_CLASSE].

● Structure du fichier CONDITION : ST_BD_CONDITION

- Nom_condition = string [MX_ID_CONDITION].
- Description = string [MX_LG_DESCRIPTION].
- Type_clause = entier (2).

● Structure du fichier COND_DECLENCHE : ST_BD_COND_DECLENCHE

- Evaluation = entier (2).
- Nom_condition = string [MX_ID_CONDITION].
- Fin_sous_schéma = string [MX_ID_SOUS_SCHEMA].
- Type_concept_déclenché = entier (2).
- Id_concept_déclenché = string [MX_ID_CONCEPT].

● Structure du fichier CLAUSE : ST_BD_CLAUSE

- Nom_condition = string [MX_ID_CONDITION].
- Champ variable composé de :

  ° Attribut : structure composée de :

- Nom_attribut = string [MX_LG_ATTRIBUT].
- Cst_alpha_numérique = string [MX_LG_CST].
- Opérateur = string [MX_LG_OPERATEUR].
- Nom_variable = string [MX_LG_VARIABLE].
- Classe = string [MX_LG_CLASSE].
- Type = string [MX_LG_TYPE].
- Nom = string [MX_LG_NOM].
- Num_copie = entier (2).

° Appartenance : structure composée de :

- Ois_nom_variable = string [MX_LG_VARIABLE].
- Ois_classe = string [MX_LG_CLASSE].
- Ois_type = string [MX_LG_TYPE].
- Ois_nom = string [MX_LG_NOM].
- Oi_nom_variable = string [MX_LG_VARIABLE].
- Oi_classe = string [MX_LG_CLASSE].
- Oi_type = string [MX_LG_TYPE].
- Oi_nom = string [MX_LG_NOM].
- Oi_num_copie = entier (2).

° Paramètre : structure composée de :

- Nom_paramètre = string [MX_LG_PARAMETRE].
- Opérateur = string [MX_LG_OPERATEUR].
- Cst_alpha_numérique = string [MX_LG_CST].

• Structure du fichier OP_PRIMITIVE : ST_BD_OP_PRIMITIVE

- Nom_opération = string [MX_LG_OPERATION].
- Description = string [MX_LG_DESCRIPTION].
- Fin_sous_schéma = string [MX_ID_SOUS_SCHEMA].
- Type_opération = entier (2).
- Type_concept_déclenché = entier (2).
- Id_concept_déclenché = string [MX_ID_CONCEPT].

• Structure du fichier OPERATION : ST_BD_OPERATION

- Nom_opération = string [MX_ID_OPERATION].
- Constituant_réception = entier (2).
- Information : champ variable composé de :

° Créer = structure ST_CREER.
° Reproduire = structure ST_REPRODUIRE_VAR.
° Recevoir = structure ST_RECEVOIR.
° Communiquer = structure ST_COMMUNIQUER_VAR.

° Consulter = structure ST_CONSULTER_VAR.
° Feuilleter = structure ST_FEUILLETER_VAR.
° Enlever = structure ST_ENLEVER_VAR.
° Modifier = structure ST_MODIFIER_VAR.
° Compléter = structure ST_COMPLETER_VAR.
° Vérifier = structure ST_VERIFIER_VAR.
° Détruire = structure ST_DETRUIRE_VAR.
° Archiver = structure ST_ARCHIVER_VAR.
° Trier = structure ST_TRIER_VAR.
° Classer = structure ST_CLASSER_VAR.
° Ranger = structure ST_RANGER_VAR.
° Replacer = structure ST_REPLACER_VAR.
° Décider = structure ST_DECIDER_VAR.
° Const_réception = structure ST_CONSTITUANT_RECEPTION

L'item CONSTITUANT_RECEPTION est utilisé pour distinguer les articles : si la valeur de l'item est FAUX alors il s'agit d'un article décrivant une opération primitive sinon si la valeur est VRAI alors il s'agit d'un constituant se rapportant à une opération RECEVOIR.

Cet artifice a été utilisé pour limiter le nombre de fichier ouvert simultanément.

Annexe 5
Description des structures de données internes

ANNEXE 5 : DESCRIPTION DES STRUCTURES DE DONNEES INTERNES

Nous allons décrire dans cette annexe les structures de données internes. Il s'agira des listes qui permettront de mémoriser dans le système des sous-schémas, des boucles, des variables et des paramètres.

L'insertion dans chacune de ces listes se fait en tête de liste. Cela veut dire que le dernier élément inséré dans une liste se trouve en première position dans cette liste.

Cette philosophie de gestion de liste a été choisie car dans la plupart des cas, on travaille sur le dernier élément inséré dans une liste. Par l'insertion en tête de liste, on optimise donc le temps de recherche dans la liste. De plus, cette option simplifie la manipulation des pointeurs en cas d'insertion ou suppression.

• LISTE DES SOUS-SCHEMAS

a) Représentation graphique :

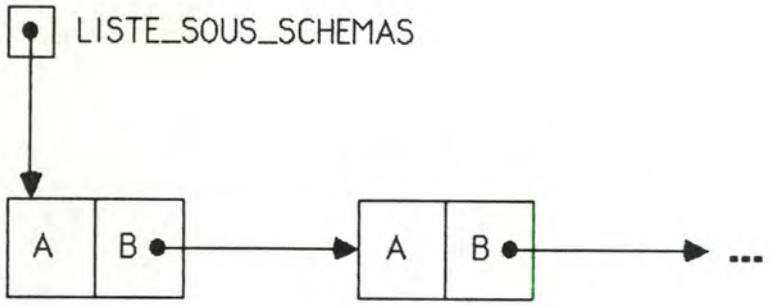


Fig A5.1. : Liste des sous-schémas

b) Description de la structure de données :

Un pointeur général **LISTE_SOUS_SCHEMAS** pointe vers le premier élément de la liste.

Chaque élément de la liste est composé de deux champs. Le premier champ (**A**) est l'identifiant du sous-schéma (format : string [MX_ID_SOUS_SCHEMA]). Le second champ (**B**) est un pointeur vers l'élément suivant de la liste.

• LISTE DES BOUCLES

a) Représentation graphique :

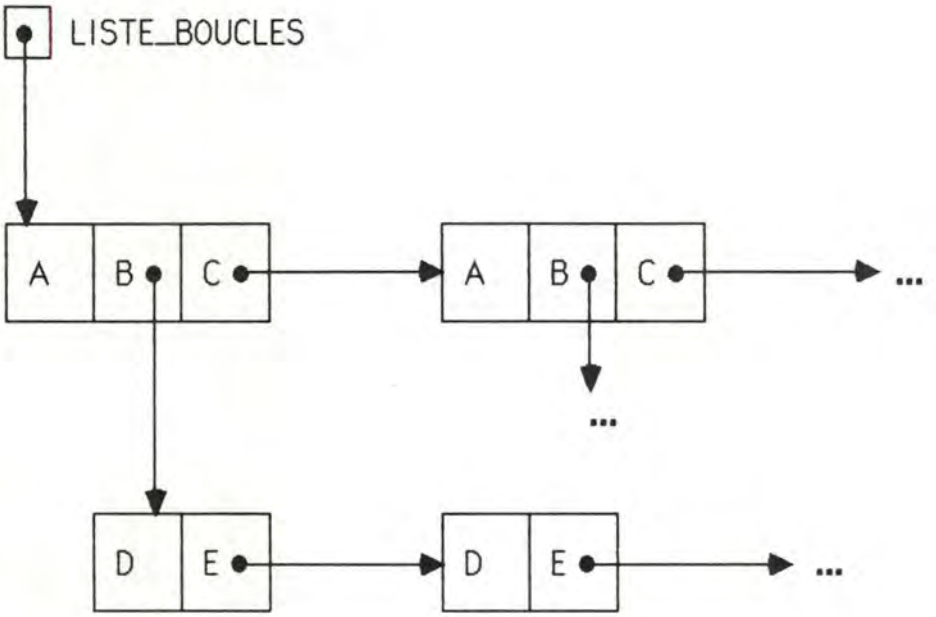


Fig. A5.2. : Liste des boucles

b) Description de la structure de données :

Un pointeur général **LISTE_BOUCLES** pointe vers le premier élément de la liste.

Chaque élément de la liste est composé de trois champs. Le premier champ (**A**) est l'identifiant de la boucle (format : string [MX_ID_BOUCLE]). Le deuxième champ (**B**) est un pointeur vers la liste des objets informationnels associée à la boucle. Le troisième champ (**C**) est un pointeur vers l'élément suivant de la liste des boucles.

Chaque élément de la liste des objets informationnels associée à une boucle est composé de deux champs. Le premier champ (**D**) est l'identifiant de l'objet informationnel (format : structure ST_IDENTIFIANT_OI). Le second champ (**E**) est un pointeur vers l'élément suivant de la liste des objets.

• LISTE DES VARIABLES

a) Représentation graphique :

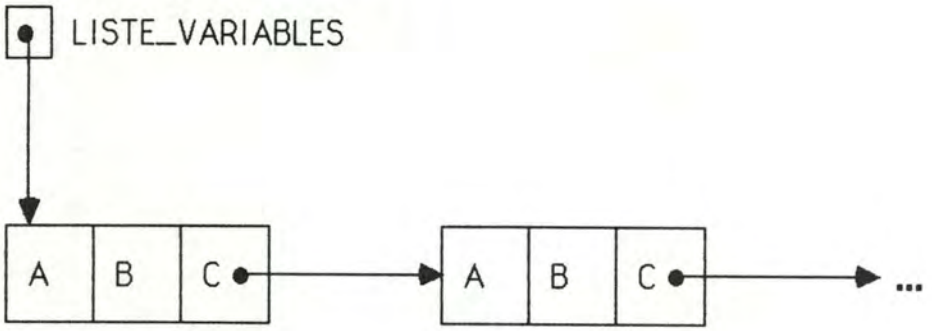


Fig A5.3 : Liste des variables

b) Description de la structure de données :

Un pointeur général **LISTE_VARIABLES** pointe vers le premier élément de la liste.

Chaque élément de la liste est composé de trois champs. Le premier champ (**A**) est l'identifiant de la variable (format : string [MX_LG_VARIABLE]). Le deuxi- ème champ (**B**) est la valeur de la variable (format : structure ST_IDENTIFIANT _OI). Le troisième (**C**) est un pointeur vers l'élément suivant de la liste.

• LISTE DES PARAMETRES

a) Représentation graphique :

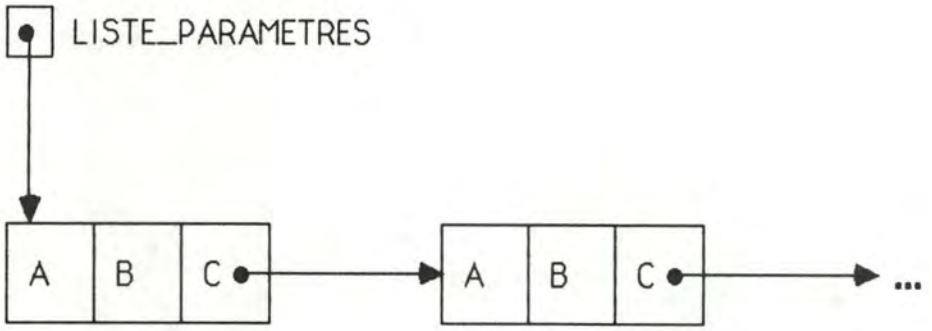


Fig A5.4 : Liste des paramètres

b) Description de la structure de données :

Un pointeur général **LISTE_PARAMETRES** pointe vers le premier élément de la liste.

Chaque élément de la liste est composé de trois champs. Le premier champ (**A**) est l'identifiant du paramètre (format : string [MX_LG_PARAMETRE]). Le deuxi- ème champ (**B**) est la valeur du paramètre (format : string [MX_LG_VALEUR_PARAMETRE]). Le troisième (**C**) est un pointeur vers l'élément suivant de la liste.

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Animation graphique de tâches de bureau

- Annexes (2ème partie) -

Mémoire présenté par
Marc Simoens et Olivier Van Pevenaeyge
en vue de l'obtention du titre de
Licencié et Maître en Informatique.

Promoteur : Mr. Roland Lesuisse.

Année académique 1986 - 1987.

TABLE DES MATIERES

Annexe 6
Programmes

ANNEXE 6 : PROGRAMMES

Se trouvent dans cette annexe les programmes de notre implémentation. L'ordre des programmes est le même que pour les spécifications externes présentées à l'annexe 3.

En raison d'impératifs horaires concernant la duplication de ce mémoire, nous n'avons pu joindre les listings concernant le module gérant le graphisme.

L'annexe qui suit représente donc l'implémentation complète du module SEQUENCEUR et l'implémentation partielle du module SIMULATION.

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : define.h                                                        */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient la definition des constantes communes */
/*               a tous les modules. (cfr. Annexe 2)                         */
/*                                                                           */
/*****************************************************************************/

#define MX_ID_CONCEPT           20

#define MX_ID_TACHE             20
#define MX_ID_SOUS_SCHEMA       20
#define MX_ID_BOUCLE            20
#define MX_ID_CONDITION         20
#define MX_ID_OPERATION         20

#define MX_TY_OPERATION         15

#define MX_LG_DESCRIPTION       80
#define MX_LG_ATTRIBUT          10
#define MX_LG_CLASSE            10
#define MX_LG_TYPE              10
#define MX_LG_NOM               10
#define MX_LG_ETAT              10
#define MX_LG_ECHEANCE          10
#define MX_LG_ORDRE             20
#define MX_LG_ETAT_CONTENU      10

#define MX_LG_PARAMETRE         20
#define MX_LG_VALEUR_PARAMETRE  15

#define MX_LG_VARIABLE          20

#define MX_OBJET_CLASSE         20
#define MX_OBJET_NOM            20

#define MX_OR_CLASSE            10
#define MX_OR_NOM               15

#define MX_OI_CLASSE            10
#define MX_OI_NOM               15

#define MX_OT_CLASSE            20
#define MX_OT_NOM               20

#define MX_LG_MODE               5

#define MX_LG_DESIR              4

#define MX_LG_OPERATEUR          2

#define MX_LG_CST               20

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

```
#define OK                          0

/*****************************************************************************/

/* definition des codes d'erreur. */

#define ERREUR                      1
#define ERREUR_SYSTEME              2
#define ERREUR_SOUS_SCHEMA          3
#define ERREUR_FIN_SOUS_SCHEMA      4
#define ERREUR_BOUCLE               5
#define ENREG_SOUS_SCHEMA           6
#define ENREG_BOUCLE                7
#define ENREG_OBJET                 8
#define ENREG_VARIABLE              9
#define SUPPR_SOUS_SCHEMA          10
#define SUPPR_BOUCLE               11
#define SUPPR_OBJET                12
#define SUPPR_VARIABLE             13
#define SUPPR_PARAMETRE            14
#define LECTURE_OBJET              15
#define CLAUSE_ATTRIBUT            16
#define CLAUSE_APPARTENANCE        17
#define OIE_CONDITION              18
#define OIS_CONDITION              19
#define OIS_BOUCLE                 20
#define OI_CREER                   21
#define OIE_REPRODUIRE             22
#define OI_RECEVOIR                23
#define OI_COMMUNIQUER             24
#define OIE_CONSULTER              25
#define OIS_FEUILLETER             26
#define OI_ENLEVER                 27
#define OI_MODIFIER                28
#define OIE_COMPLETER              29
#define OI_VERIFIER                30
#define OI_DETRUIRE                31
#define OI_ARCHIVER                32
#define OIS_TRIER                  33
#define OI_CLASSER                 34
#define OI_RANGER                  35
#define OI_REPLACER                36
#define OI_DECIDER                 37
#define VAR_EXISTE_PAS             38
#define PAR_EXISTE_PAS             39
#define PHOTOCOPIEUSE_EXISTE_PAS   40
#define BOITE_IN_EXISTE_PAS        41
#define BOITE_OUT_EXISTE_PAS       42
#define BOITE_ARCHIVE_EXISTE_PAS   43
#define TELEPHONE_EXISTE_PAS       44
#define POUBELLE_EXISTE_PAS        45
#define CLASSEMENT_EXISTE_PAS      46
#define RANGEMENT_EXISTE_PAS       47
#define PAS_SUR_OUTIL_TRAVAIL      48
#define NUM_DEPART_INCORRECT       49
#define ERREUR_FICHIER             50
```

```
#define CLASSE_REPRODUIRE       51
#define CLASSE_COMMUNIQUER      52
#define CLASSE_CONSULTER        53
#define CLASSE_FEUILLETER       54
#define CLASSE_ENLEVER          55
#define CLASSE_MODIFIER         56
#define CLASSE_COMPLETER        57
#define CLASSE_VERIFIER         58
#define CLASSE_DETRUIRE         59
#define CLASSE_ARCHIVER         60
#define CLASSE_TRIER            61
#define CLASSE_CLASSER          62
#define CLASSE_DECIDER          63

#define EXISTE_PAS              69
#define OI_EXISTE_PAS           70
#define OIS_EXISTE_PAS          71
#define PILE_VIDE               72
#define PILE_EXISTE_PAS         73

#define ERREUR_BD_OUVERTURE     100
#define ERREUR_BD_FERMETURE     101

/*************************************************************************/

/* defintion des codes de retour. */

#define VRAI                    1
#define FAUX                    2
#define PRESENT                 3
#define ABSENT                  4
#define DERNIER                 5
#define VIDE                    6
#define SUITE                   7
#define FIN_TACHE               8
#define FIN_SOUS_SCHEMA         9
#define FIN_BOUCLE              10
#define BOUCLE_NEXT             11
#define TROISIEME_CRITERE       12
#define FICHIER_VIDE            13
#define INFERIEUR               14
#define CLASSEMENT              15
#define NON_CLASSEMENT          16
#define TABLE_DE_TRAVAIL        17
#define PHOTOCOPIEUSE           18

/*************************************************************************/

/* definition des differents types de concept */

#define TACHE                   1
#define SOUS_SCHEMA             2
#define BOUCLE                  3
#define CONDITION               4
#define OPERATION_PRIMITIVE     5
```

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

/* definition des differents types de critere */

#define CR_CONSTITUANT          1
#define CR_CLASSE               2
#define CR_TOUS                 3

/*****************************************************************************/

/* definition des differents types de clause */

#define CL_ATTRIBUT             1
#define CL_APPARTENANCE         2
#define CL_PARAMETRE            3

/*****************************************************************************/

/* defintion des differents types d'objets */

#define OBJET_RANGEMENT         1
#define OBJET_INTERFACE         2
#define OUTIL_TRAVAIL           3

/*****************************************************************************/

/* defintion generales */

#define AND                     &&
#define OR                      ||
#define TRUE                     1
#define FALSE                    0
#define BOOLEAN                 int

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : type.h                                                          */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient la definition de structures communes    */
/*               a tous les modules. (cfr. Annexe 2)                         */
/*                                                                           */
/*****************************************************************************/

typedef struct liste_identifiant_oi
              { char classe          [MX_LG_CLASSE + 1];
                char type            [MX_LG_TYPE + 1];
                char nom             [MX_LG_NOM + 1];
                int  num_copie;
                struct liste_identifiant_oi *ptr_suivant;
              }
       st_liste_identifiant_oi;

/*****************************************************************************/

typedef struct liste_valeur_oi
              { char classe          [MX_LG_CLASSE + 1];
                char type            [MX_LG_TYPE + 1];
                char nom             [MX_LG_NOM + 1];
                int  num_copie;
                char etat            [MX_LG_ETAT + 1];
                char echeance        [MX_LG_ECHEANCE + 1];
                struct liste_valeur_oi *ptr_suivant;
              }
       st_liste_valeur_oi;

/*****************************************************************************/

typedef struct identifiant_oi
              { char classe          [MX_LG_CLASSE + 1];
                char type            [MX_LG_TYPE + 1];
                char nom             [MX_LG_NOM + 1];
                int  num_copie;
              }
       st_identifiant_oi;

/*****************************************************************************/

typedef struct valeur_oi
              { char classe          [MX_LG_CLASSE + 1];
                char type            [MX_LG_TYPE + 1];
                char nom             [MX_LG_NOM + 1];
                int  num_copie;
                char etat            [MX_LG_ETAT + 1];
                char echeance        [MX_LG_ECHEANCE + 1];
                char ordre_clas      [MX_LG_ORDRE + 1];
                char etat_contenu    [MX_LG_ETAT_CONTENU + 1];
              }
       st_valeur_oi;

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

```
typedef struct { char nom_operation  [MX_ID_OPERATION + 1];
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 st_identifiant_oi   identifiant_oi;
               }
        st_constituant_reception;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char ordre_clas     [MX_LG_ORDRE + 1];
               }
        st_creer;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 int  nbr_copie;
                 int  num_depart;
               }
        st_reproduire;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 int  nbr_copie;
                 int  num_depart;
               }
        st_reproduire_var;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char ordre_clas     [MX_LG_ORDRE + 1];
                 char mode_reception [MX_LG_MODE + 1];
                 st_liste_valeur_oi  *composants;
               }
        st_recevoir;

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

```
typedef struct { char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat            [MX_LG_ETAT + 1];
                 char echeance        [MX_LG_ECHEANCE + 1];
                 char mode_communication [MX_LG_MODE + 1];
               }
       st_communiquer;

/*****************************************************************************/

typedef struct { char nom_variable    [MX_LG_VARIABLE + 1];
                 char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat            [MX_LG_ETAT + 1];
                 char echeance        [MX_LG_ECHEANCE + 1];
                 char mode_communication [MX_LG_MODE + 1];
               }
       st_communiquer_var;

/*****************************************************************************/

typedef struct { char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat            [MX_LG_ETAT + 1];
                 char echeance        [MX_LG_ECHEANCE + 1];
               }
       st_consulter;

/*****************************************************************************/

typedef struct { char nom_variable    [MX_LG_VARIABLE + 1];
                 char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat            [MX_LG_ETAT + 1];
                 char echeance        [MX_LG_ECHEANCE + 1];
               }
       st_consulter_var;

/*****************************************************************************/

typedef struct { char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 char etat            [MX_LG_ETAT + 1];
                 char echeance        [MX_LG_ECHEANCE + 1];
                 char ordre_feuilletage [MX_LG_ORDRE + 1];
               }
       st_feuilleter;
```

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

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char ordre_feuilletage [MX_LG_ORDRE + 1];
               }
       st_feuilleter_var;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char nom_pile       [MX_LG_NOM + 1];
               }
       st_enlever;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char nom_pile       [MX_LG_NOM + 1];
                 char variable_constituant [MX_LG_VARIABLE + 1];
               }
       st_enlever_var;

/*****************************************************************************/

typedef struct { char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char nom_message    [MX_LG_NOM + 1];
                 int  num_copie_message;
               }
       st_modifier;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char nom_message    [MX_LG_NOM + 1];
                 int  num_copie_message;
               }
       st_modifier_var;

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

```
typedef struct { char type          [MX_LG_TYPE + 1];
                 char nom           [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat          [MX_LG_ETAT + 1];
                 char echeance      [MX_LG_ECHEANCE + 1];
               }
       st_completer;

/*****************************************************************************/

typedef struct { char nom_variable  [MX_LG_VARIABLE + 1];
                 char type          [MX_LG_TYPE + 1];
                 char nom           [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat          [MX_LG_ETAT + 1];
                 char echeance      [MX_LG_ECHEANCE + 1];
               }
       st_completer_var;

/*****************************************************************************/

typedef struct { char classe        [MX_LG_CLASSE + 1];
                 char type          [MX_LG_TYPE + 1];
                 char nom           [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat          [MX_LG_ETAT + 1];
                 char echeance      [MX_LG_ECHEANCE + 1];
                 char nom_parametre [MX_LG_PARAMETRE + 1];
               }
       st_verifier;

/*****************************************************************************/

typedef struct { char nom_variable  [MX_LG_VARIABLE + 1];
                 char classe        [MX_LG_CLASSE + 1];
                 char type          [MX_LG_TYPE + 1];
                 char nom           [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat          [MX_LG_ETAT + 1];
                 char echeance      [MX_LG_ECHEANCE + 1];
                 char nom_parametre [MX_LG_PARAMETRE + 1];
               }
       st_verifier_var;

/*****************************************************************************/

typedef struct { char classe        [MX_LG_CLASSE + 1];
                 char type          [MX_LG_TYPE + 1];
                 char nom           [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat          [MX_LG_ETAT + 1];
                 char echeance      [MX_LG_ECHEANCE + 1];
               }
       st_detruire;

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

```
typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
               }
       st_detruire_var;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
               }
       st_archiver;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
               }
       st_archiver_var;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char ordre_tri      [MX_LG_ORDRE + 1];
               }
       st_trier;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char etat           [MX_LG_ETAT + 1];
                 char echeance       [MX_LG_ECHEANCE + 1];
                 char ordre_tri      [MX_LG_ORDRE + 1];
               }
       st_trier_var;
```

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

typedef struct { char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char dest_classe     [MX_LG_CLASSE + 1];
                 char dest_type       [MX_LG_TYPE + 1];
                 char dest_nom        [MX_LG_NOM + 1];
               }
       st_classer;

/*****************************************************************************/

typedef struct { char nom_variable    [MX_LG_VARIABLE + 1];
                 char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char dest_classe     [MX_LG_CLASSE + 1];
                 char dest_type       [MX_LG_TYPE + 1];
                 char dest_nom        [MX_LG_NOM + 1];
               }
       st_classer_var;

/*****************************************************************************/

typedef struct { char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char dest_classe     [MX_OR_CLASSE + 1];
                 char dest_nom        [MX_OR_NOM + 1];
               }
       st_ranger;

/*****************************************************************************/

typedef struct { char nom_variable    [MX_LG_VARIABLE + 1];
                 char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char dest_classe     [MX_OR_CLASSE + 1];
                 char dest_nom        [MX_OR_NOM + 1];
               }
       st_ranger_var;

/*****************************************************************************/

typedef struct { char classe          [MX_LG_CLASSE + 1];
                 char type            [MX_LG_TYPE + 1];
                 char nom             [MX_LG_NOM + 1];
                 int  num_copie;
                 char gr_classe       [MX_LG_CLASSE + 1];
                 char gr_type         [MX_LG_TYPE+ 1];
```

```
                }
        st_replacer;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char gr_classe      [MX_LG_CLASSE + 1];
                 char gr_type        [MX_LG_TYPE + 1];
               }
        st_replacer_var;

/*****************************************************************************/

typedef struct { char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char nom_parametre  [MX_LG_PARAMETRE + 1];
               }
        st_decider;

/*****************************************************************************/

typedef struct { char nom_variable   [MX_LG_VARIABLE + 1];
                 char classe         [MX_LG_CLASSE + 1];
                 char type           [MX_LG_TYPE + 1];
                 char nom            [MX_LG_NOM + 1];
                 int  num_copie;
                 char nom_parametre  [MX_LG_PARAMETRE + 1];
               }
        st_decider_var;

/*****************************************************************************/

typedef struct { char type_operation [MX_TY_OPERATION + 1];
                 char nom_operation  [MX_ID_OPERATION + 1];
                 char description    [MX_LG_DESCRIPTION + 1];
                 union { st_creer           creer;
                         st_reproduire      reproduire;
                         st_recevoir        recevoir;
                         st_communiquer     communiquer;
                         st_consulter       consulter;
                         st_feuilleter      feuilleter;
                         st_enlever         enlever;
                         st_modifier        modifier;
                         st_completer       completer;
                         st_verifier        verifier;
                         st_detruire        detruire;
                         st_archiver        archiver;
                         st_trier           trier;
                         st_classer         classer;
                         st_ranger          ranger;
                         st_replacer        replacer;
```

```
                        st_decider          decider;
                      }
                information;
              }
        st_operation;

/*****************************************************************************/

typedef struct { char type_operation [MX_TY_OPERATION + 1];
                 char nom_operation  [MX_ID_OPERATION + 1];
                 char description    [MX_LG_DESCRIPTION + 1];
                 union { st_creer            creer;
                         st_reproduire_var   reproduire;
                         st_recevoir         recevoir;
                         st_communiquer_var  communiquer;
                         st_consulter_var    consulter;
                         st_feuilleter_var   feuilleter;
                         st_enlever_var      enlever;
                         st_modifier_var     modifier;
                         st_completer_var    completer;
                         st_verifier_var     verifier;
                         st_detruire_var     detruire;
                         st_archiver_var     archiver;
                         st_trier_var        trier;
                         st_classer_var      classer;
                         st_ranger_var       ranger;
                         st_replacer_var     replacer;
                         st_decider_var      decider;
                       }
                 information;
               }
        st_operation_var;

/*****************************************************************************/

typedef struct { int     type_concept;
                 char    id_concept_declencheur [MX_ID_CONCEPT + 1];
                 union { int condition;
                         int boucle;
                       }
                 critere;
               }
        st_c_declencheur;

/*****************************************************************************/

typedef struct { int     type_concept;
                 char    id_concept_declenche [MX_ID_CONCEPT + 1];
               }
        st_c_declenche;

/*****************************************************************************/

typedef struct { char    nom_variable [MX_LG_VARIABLE + 1];
                 int     type_critere;
                 union { struct { char classe   [MX_LG_CLASSE + 1];
                                  char type     [MX_LG_TYPE + 1];
```

```
                                char nom       [MX_LG_NOM + 1];
                                char desir     [MX_LG_DESIR + 1];
                                char variable [MX_LG_VARIABLE + 1];
                              }
                        critere_constituant;
                        struct { char classe   [MX_LG_CLASSE + 1];
                              }
                        critere_classe;
                      }
                information;
              }
        st_critere_boucle;

/*************************************************************************/

typedef struct { int     type_clause;
                 union { struct { char nom_attribut     [MX_LG_ATTRIBUT + 1];
                                  char cst_alpha_num    [MX_LG_CST + 1];
                                  char operateur        [MX_LG_OPERATEUR + 1];
                                  char nom_variable     [MX_LG_VARIABLE + 1];
                                  char classe           [MX_LG_CLASSE + 1];
                                  char type             [MX_LG_TYPE + 1];
                                  char nom              [MX_LG_NOM + 1];
                                  int  num_copie;
                                }
                         clause_attribut;
                         struct { char oi_nom_variable  [MX_LG_VARIABLE + 1];
                                  char oi_classe        [MX_LG_CLASSE + 1];
                                  char oi_type          [MX_LG_TYPE + 1];
                                  char oi_nom           [MX_LG_NOM + 1];
                                  int  oi_num_copie;
                                  char ois_nom_variable [MX_LG_VARIABLE + 1];
                                  char ois_classe       [MX_LG_CLASSE + 1];
                                  char ois_type         [MX_LG_TYPE + 1];
                                  char ois_nom          [MX_LG_NOM + 1];
                                }
                         clause_appartenance;
                         struct { char nom_parametre    [MX_LG_PARAMETRE + 1];
                                  char cst_alpha_num    [MX_LG_CST + 1];
                                  char operateur        [MX_LG_OPERATEUR + 1];
                                }
                         clause_parametre;
                       }
                 information;
                 }
        st_clause_condition;

/*************************************************************************/

typedef short int ty_code_retour;

/*************************************************************************/

typedef short int ty_code_erreur;

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : debug.h                                                         */
/*                                                                           */
/* DESCRIPTION : Ce fichier indique les fichiers pour lesquels une "trace" */
/*               est necessaire.                                             */
/*                                                                           */
/*****************************************************************************/

#define DEBUG_MAIN              1
#define DEBUG_SIMULATION        1
#define DEBUG_SEQUENCEUR        1
#define DEBUG_CONCEPT_SUIVANT   1
#define DEBUG_SOUS_SCHEMA       1
#define DEBUG_FIN_SOUS_SCHEMA   1
#define DEBUG_FIN_TACHE         1
#define DEBUG_BOUCLE            1
#define DEBUG_CONDITION         1
#define DEBUG_OP_PRIMITIVE      1
#define DEBUG_VARIABLE          1
#define DEBUG_PARAMETRE         1
#define DEBUG_MEMO_SOUS_SCHEMA  1
#define DEBUG_MEMO_BOUCLE       1
#define DEBUG_CONTROLES_A_PRIORI 1
#define DEBUG_GRAPHIQUE         1
#define DEBUG_MANIPULATION_BD   1
#define DEBUG_MESSAGE_ERREUR    1
#define DEBUG_BASE_DE_DONNEES   1

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : main.c                                                          */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le programme principal.                 */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

main(argc,argv)

int  argc;
char *argv[];

{  char identifiant_tache [MX_ID_TACHE + 1];

#if DEBUG_MAIN
  printf(">>>MAIN.\n");
#endif

   if (argc == 2)
   {  strncpy (identifiant_tache,argv[1],MX_ID_TACHE);
      identifiant_tache [MX_ID_TACHE] = '\0';

      simulation (identifiant_tache);
   }

#if DEBUG_MAIN
  printf("<<<MAIN.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : simulation.c                                                    */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de simulation.                */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

simulation (identifiant_tache)

char identifiant_tache [MX_ID_TACHE + 1];

{  st_operation   op_primitive;
   ty_code_retour code_retour;
   ty_code_erreur code_erreur;

#if DEBUG_SIMULATION
  printf(">>>SIMULATION.\n");
  printf("-----------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_tache : %s.\n",identifiant_tache);
  printf("-----------------------------------------------------\n");
#endif

ouverture_bd (&code_erreur);

if (code_erreur == OK)
   sequenceur_tache (identifiant_tache,&op_primitive,&code_retour,
                     &code_erreur);
   else message_erreur (ERREUR_BD_OUVERTURE);

while (code_retour == OK AND code_erreur == OK)
      {
      controles_a_priori (&op_primitive,&code_retour,&code_erreur);

      if (code_erreur == OK)
         manipulation_bd (&op_primitive,&code_retour,&code_erreur);

      if (code_erreur == OK)
         graphique (&op_primitive,&code_retour,&code_erreur);

      if (code_erreur == OK)
         sequenceur_op_prim (op_primitive.nom_operation,&op_primitive,
                             &code_retour,&code_erreur);
      }

fermeture_bd (&code_erreur);

if (code_erreur == ERREUR_BD_FERMETURE)
   message_erreur (code_erreur);
```

```
#if DEBUG_SIMULATION
  printf("<<<SIMULATION.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : sequenceur.c                                                    */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion du sequencement.   */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

sequenceur_tache (identifiant_tache,op_primitive,code_retour,code_erreur)

char           identifiant_tache [MX_ID_TACHE + 1];
st_operation   *op_primitive;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{
short            fin;
short            type_concept;
char             identifiant_concept [MX_ID_CONCEPT + 1];
char             identifiant_boucle [MX_ID_BOUCLE + 1];
char             identif [MX_ID_CONCEPT + 1];
st_c_declencheur concept_declencheur;
st_c_declenche   concept_declenche;
ty_code_retour   cr, cr1;
ty_code_erreur   ce, ce1;

/*****************************************************************************
 * DEBUT DU PROGRAMME                                                        *
 *****************************************************************************/

#if DEBUG_SEQUENCEUR
  printf(">>>SEQUENCEUR_TACHE.\n");
  printf("-------------------------------------------------------\n");
  printf("Argument :\n");
  printf("Identifiant_tache : %s\n",identifiant_tache);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

fin          = FALSE;
type_concept = TACHE;
strcpy (identifiant_concept, identifiant_tache);

initialisation_boucle ();
initialisation_variable ();
initialisation_parametre ();
initialisation_sous_schema ();
```

```
/*****************************************************************************
 * BOUCLAGE JUSQU'A TROUVE L'OPERATION PRIMITIVE SI ELLE EXISTE              *
 *****************************************************************************/

while (fin == FALSE AND *code_erreur == OK AND *code_retour == OK)
      {
      concept_declencheur.type_concept = type_concept;
      strcpy (concept_declencheur.id_concept_declencheur,identifiant_concept);
      concept_suivant (&concept_declencheur, &concept_declenche, &cr, &ce);

/*****************************************************************************
 * LE CONCEPT DECLENCHE EST LA FIN D'UN SOUS-SCHEMA                           *
 *****************************************************************************/

      if (cr == FIN_SOUS_SCHEMA)
         {

         fin_sous_schema (concept_declenche.id_concept_declenche, &cr1, &ce1);

         *code_erreur = ce1;
         if (*code_erreur == OK)
            switch (cr1)
                  {
                  case FIN_TACHE :
                       *code_retour = FIN_TACHE;
                       fin_tache (&ce1);
                       *code_erreur = ce1;
                       break;
                  case BOUCLE_NEXT :
                       boucle_next (identifiant_boucle, &cr1, &ce1);
                       *code_erreur = ce1;

                       switch (cr1)
                             {
                             case FIN_BOUCLE :
                                  concept_declencheur.critere.boucle =SUITE;
                                  break;
                             case OK :
                                  concept_declencheur.critere.boucle =
                                  SOUS_SCHEMA;
                                  break;
                             }

                       strcpy (identifiant_concept, identifiant_boucle);
                       type_concept = BOUCLE;
                       break;
                  }
         }

      if (cr == OK)
         switch (concept_declenche.type_concept)
               {
/*****************************************************************************
 * LE CONCEPT DECLENCHE EST UNE BOUCLE                                       *
 *****************************************************************************/
               case BOUCLE :
```

```
                    boucle_first (concept_declenche.id_concept_declenche,
                                  &cr1, &ce1);

                    *code_erreur = ce1;
                    if (*code_erreur == OK)
                       switch (cr1)
                              {
                              case FIN_BOUCLE :
                                   concept_declencheur.critere.boucle =SUITE;
                                   strcpy (identifiant_concept,
                                     concept_declenche.id_concept_declenche);
                                   type_concept = BOUCLE;
                                   break;
                              case OK :
                                   concept_declencheur.critere.boucle =
                                   SOUS_SCHEMA;
                                   strcpy (identifiant_concept,
                                     concept_declenche.id_concept_declenche);
                                   type_concept = BOUCLE;
                                   break;
                              }
                    break;

/*************************************************************************
 * LE CONCEPT DECLENCHE EST UNE OPERATION PRIMITIVE                      *
 *************************************************************************/

               case OPERATION_PRIMITIVE :
                    fin = TRUE;
                    strcpy (identif,concept_declenche.id_concept_declenche);

                    operation_primitive (identif,op_primitive,
                                         &cr1,&ce1);

                    *code_erreur = ce1;
                    break;

/*************************************************************************
 * LE CONCEPT DECLENCHE EST UNE CONDITION                                *
 *************************************************************************/

               case CONDITION :

                    condition (concept_declenche.id_concept_declenche, &cr1,
                               &ce1);

                    *code_erreur = ce1;
                    if (*code_erreur == OK)
                       {
                       switch (cr1)
                              {
                              case VRAI :
                                  concept_declencheur.critere.condition=VRAI;
                                  break;
                              case FAUX :
                                  concept_declencheur.critere.condition=FAUX;
                                  break;
```

```
                        }
                    strcpy (identifiant_concept, concept_declenche.
                            id_concept_declenche);
                    type_concept = CONDITION;
                    }
                break;

/*****************************************************************************
 * LE CONCEPT DECLENCHE EST UN SOUS-SCHEMA                                   *
 *****************************************************************************/

            case SOUS_SCHEMA :

                sous_schema (concept_declenche.id_concept_declenche,
                             &cr1, &ce1);

                *code_erreur = ce1;
                if (*code_erreur == OK)
                    {
                    strcpy (identifiant_concept,
                            concept_declenche.id_concept_declenche);
                    type_concept = SOUS_SCHEMA;
                    }
                break;
            }
        }

/*****************************************************************************
 * IMPRESSION D'UN MESSAGE D'ERREUR SI UNE ERREUR A ETE DETECTEE             *
 *****************************************************************************/

if (*code_erreur != OK)
    {
    message_erreur (*code_erreur);

    *code_erreur = ERREUR;
    }


#if DEBUG_SEQUENCEUR
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  printf("Nom de l'operation declenchee = %s.\n",op_primitive->nom_operation);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SEQUENCEUR_TACHE.\n");
#endif
}

/*****************************************************************************/

sequenceur_op_prim (identifiant_op_prim,op_primitive,code_retour,
                    code_erreur)

char            identifiant_op_prim [MX_ID_OPERATION + 1];
st_operation    *op_primitive;
```

```
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{
short             fin;
short             type_concept;
char              identifiant_concept [MX_ID_CONCEPT + 1];
char              identifiant_boucle [MX_ID_BOUCLE + 1];
char              identif [MX_ID_CONCEPT + 1];
st_c_declencheur concept_declencheur;
st_c_declenche   concept_declenche;
ty_code_retour   cr, cr1;
ty_code_erreur   ce, ce1;

/*****************************************************************************
 * DEBUT DU PROGRAMME                                                        *
 *****************************************************************************/

#if DEBUG_SEQUENCEUR
  printf(">>>SEQUENCEUR_OP_PRIM.\n");
  printf("-----------------------------------------------------\n");
  printf("Argument :\n");
  printf("Operation primitive en entree : %s.\n", identifiant_op_prim);
  printf("-----------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

fin          = FALSE;
type_concept = OPERATION_PRIMITIVE;
strcpy (identifiant_concept, identifiant_op_prim);

/*****************************************************************************
 * BOUCLAGE JUSQU'A TROUVE L'OPERATION PRIMITIVE SI ELLE EXISTE              *
 *****************************************************************************/

while (fin == FALSE AND *code_erreur == OK AND *code_retour == OK)
      {
      concept_declencheur.type_concept = type_concept;
      strcpy (concept_declencheur.id_concept_declencheur,
identifiant_concept);

      concept_suivant (&concept_declencheur, &concept_declenche, &cr, &ce);

/*****************************************************************************
 * LE CONCEPT DECLENCHE EST LA FIN D'UN SOUS-SCHEMA                           *
 *****************************************************************************/

      if (cr == FIN_SOUS_SCHEMA)
         {

         fin_sous_schema (concept_declenche.id_concept_declenche, &cr1, &ce1);

         *code_erreur = ce1;
         if (*code_erreur == OK)
            switch (cr1)
```

```
                    {
                    case FIN_TACHE :
                         *code_retour = FIN_TACHE;
                         fin_tache (&ce1);
                         *code_erreur = ce1;
                         break;
                    case BOUCLE_NEXT :
                         boucle_next (identifiant_boucle, &cr1, &ce1);
                         *code_erreur = ce1;

                         switch (cr1)
                                {
                                case FIN_BOUCLE :
                                     concept_declencheur.critere.boucle =SUITE;
                                     break;
                                case OK :
                                     concept_declencheur.critere.boucle =
                                     SOUS_SCHEMA;
                                     break;
                                }

                         strcpy (identifiant_concept, identifiant_boucle);
                         type_concept = BOUCLE;
                         break;
                    }
          }

     if (cr == OK)
        switch (concept_declenche.type_concept)
               {
/*****************************************************************************
 * LE CONCEPT DECLENCHE EST UNE BOUCLE                                       *
 *****************************************************************************/
               case BOUCLE :

                    boucle_first (concept_declenche.id_concept_declenche,
                                  &cr1, &ce1);

                    *code_erreur = ce1;
                    if (*code_erreur == OK)
                       switch (cr1)
                              {
                              case FIN_BOUCLE :
                                   concept_declencheur.critere.boucle =SUITE;
                                   strcpy (identifiant_concept,
                                     concept_declenche.id_concept_declenche);
                                   type_concept = BOUCLE;
                                   break;
                              case OK :
                                   concept_declencheur.critere.boucle =
                                   SOUS_SCHEMA;
                                   strcpy (identifiant_concept,
                                     concept_declenche.id_concept_declenche);
                                   type_concept = BOUCLE;
                                   break;
                              }
                    break;
```

```
/*****************************************************************************
 * LE CONCEPT DECLENCHE EST UNE OPERATION PRIMITIVE                           *
 *****************************************************************************/

                case OPERATION_PRIMITIVE :
                     fin = TRUE;
                     strcpy (identif,concept_declenche.id_concept_declenche);

                     operation_primitive (identif,op_primitive,
                                          &cr1,&ce1);

                     *code_erreur = ce1;
                     break;

/*****************************************************************************
 * LE CONCEPT DECLENCHE EST UNE CONDITION                                     *
 *****************************************************************************/

                case CONDITION :

                     condition (concept_declenche.id_concept_declenche, &cr1,
                               &ce1);

                     *code_erreur = ce1;
                     if (*code_erreur == OK)
                        {
                        switch (cr1)
                               {
                               case VRAI :
                                   concept_declencheur.critere.condition=VRAI;
                                   break;
                               case FAUX :
                                   concept_declencheur.critere.condition=FAUX;
                                   break;
                               }
                        strcpy (identifiant_concept, concept_declenche.
                                id_concept_declenche);
                        type_concept = CONDITION;
                        }
                     break;

/*****************************************************************************
 * LE CONCEPT DECLENCHE EST UN SOUS-SCHEMA                                    *
 *****************************************************************************/

                case SOUS_SCHEMA :

                     sous_schema (concept_declenche.id_concept_declenche,
                                  &cr1, &ce1);

                     *code_erreur = ce1;
                     if (*code_erreur == OK)
                        {
                        strcpy (identifiant_concept,
                                concept_declenche.id_concept_declenche);
                        type_concept = SOUS_SCHEMA;
```

```
                        }
                    break;
                }
        }

/*****************************************************************************
 * IMPRESSION D'UN MESSAGE D'ERREUR SI UNE ERREUR A ETE DETECTEE             *
 *****************************************************************************/

if (*code_erreur != OK)
   {
   message_erreur (*code_erreur);

   *code_erreur = ERREUR;
   }

#if DEBUG_SEQUENCEUR
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  printf("Nom de l'operation declenchee : %s.\n",op_primitive->nom_operation);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SEQUENCEUR_OP_PRIM.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : cpt_suivant.c                                                   */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion du concept         */
/*               suivant a simuler.                                          */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

concept_suivant
(concept_declencheur,concept_declenche,code_retour,code_erreur)

st_c_declencheur *concept_declencheur;
st_c_declenche   *concept_declenche;
ty_code_retour   *code_retour;
ty_code_erreur   *code_erreur;

{  char id_concept [MX_ID_CONCEPT + 1];
   int  critere;

#if DEBUG_CONCEPT_SUIVANT
  printf(">>>CONCEPT_SUIVANT.\n");
  printf("-----------------------------------------------------\n");
  printf("Concept dont il faut rechercher le suivant :\n");
  printf("Type    : %d.\n",concept_declencheur->type_concept);
  printf("Nom     : %s.\n",concept_declencheur->id_concept_declencheur);
  printf("Critere : %d.\n",concept_declencheur->critere.condition);
  printf("-----------------------------------------------------\n");
#endif

   strcpy (id_concept,concept_declencheur->id_concept_declencheur);

   switch (concept_declencheur->type_concept)

   {  case TACHE : tache_concept_suivant (id_concept,concept_declenche,
                                          code_retour,code_erreur);
                   break;

      case SOUS_SCHEMA : sous_schema_concept_suivant (id_concept,
                                concept_declenche,code_retour,code_erreur);
                         break;

      case BOUCLE : critere = concept_declencheur->critere.boucle;
                    boucle_concept_suivant (id_concept,&critere,
                                concept_declenche,code_retour,code_erreur);
                    break;

      case CONDITION : critere = concept_declencheur->critere.condition;
                       condition_concept_suivant (id_concept,&critere,
                                concept_declenche,code_retour,code_erreur);
```

```
                        break;

      case OPERATION_PRIMITIVE : operation_concept_suivant (id_concept,
                                   concept_declenche,code_retour,code_erreur);
                                 break;
   }

#if DEBUG_CONCEPT_SUIVANT
  printf("-------------------------------------------------------\n");
  printf("Concept suivant declenche :\n");
  printf("Type : %d.\n",concept_declenche->type_concept);
  printf("Nom  : %s.\n",concept_declenche->id_concept_declenche);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONCEPT_SUIVANT.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : ss_schema.c                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des sous-schemas */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

sous_schema (identifiant_sous_schema,code_retour,code_erreur)

char           identifiant_sous_schema [MX_ID_SOUS_SCHEMA + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  ty_code_retour cr;
   ty_code_erreur ce;

#if DEBUG_SOUS_SCHEMA
  printf(">>>SOUS_SCHEMA.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_sous_schema : %s.\n",identifiant_sous_schema);
  printf("-------------------------------------------------------\n");
#endif

   *code_retour = OK;
   *code_erreur = OK;

   existence_sous_schema (identifiant_sous_schema, &cr, &ce);

   if (cr == PRESENT)
      *code_erreur = ERREUR_SOUS_SCHEMA;
      else {
           enregistrement_sous_schema (identifiant_sous_schema, &cr, &ce);
           if (ce == ERREUR_SYSTEME)
              *code_erreur = ENREG_SOUS_SCHEMA;
           }

#if DEBUG_SOUS_SCHEMA
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SOUS_SCHEMA.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : fin_ss_schema.c                                                 */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion de fin de          */
/*               sous-schema.                                                */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

fin_sous_schema (identifiant_sous_schema,code_retour,code_erreur)

char           identifiant_sous_schema [MX_ID_SOUS_SCHEMA + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  char           identifiant_ss [MX_ID_SOUS_SCHEMA + 1];
   ty_code_retour cr;
   ty_code_erreur ce;


#if DEBUG_FIN_SOUS_SCHEMA
  printf(">>>FIN_SOUS_SCHEMA.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_sous_schema : %s.\n",identifiant_sous_schema);
  printf("-------------------------------------------------------\n");
#endif

   *code_retour = OK;
   *code_erreur = OK;

   suppression_sous_schema (identifiant_ss, &cr, &ce);

   if (ce == ERREUR_SYSTEME)
      *code_erreur = SUPPR_SOUS_SCHEMA;
      else if (strcmp (identifiant_ss, identifiant_sous_schema) != 0)
              *code_erreur = ERREUR_FIN_SOUS_SCHEMA;
              else if (cr == DERNIER)
                      *code_retour = FIN_TACHE;
                      else *code_retour = BOUCLE_NEXT;

#if DEBUG_FIN_SOUS_SCHEMA
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<FIN_SOUS_SCHEMA.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : fin_tache.c                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion de fin de tache.   */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

fin_tache (code_erreur)

ty_code_erreur *code_erreur;

{  ty_code_erreur ce;

#if DEBUG_FIN_TACHE
  printf(">>>FIN_TACHE.\n");
#endif

   *code_erreur = OK;
   suppression_parametre (&ce);

   if (ce == ERREUR_SYSTEME)
      *code_erreur = SUPPR_PARAMETRE;
      else {
           suppression_variable (&ce);
           if (ce == ERREUR_SYSTEME)
              *code_erreur = SUPPR_VARIABLE;
           }

#if DEBUG_FIN_TACHE
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<FIN_TACHE.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : boucle.c                                                        */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des boucles.       */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

boucle_first (identifiant_boucle,code_retour,code_erreur)

char           identifiant_boucle [MX_ID_BOUCLE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{
st_liste_identifiant_oi *liste_oi[1], *ptr;
st_critere_boucle       info_critere;
st_identifiant_oi       valeur_variable, identifiant_ois, identifiant_oi;
char                    type_oi [MX_LG_DESIR + 1];
ty_code_erreur          ce;
ty_code_retour          cr;


#if DEBUG_BOUCLE
  printf(">>>BOUCLE_FIRST.\n");
  printf("-----------------------------------------------------\n");
  printf("Argument :\n");
  printf("Identifiant boucle : %s.\n",identifiant_boucle);
  printf("-----------------------------------------------------\n");
#endif

/*****************************************************************************
 *   ENREGISTRER L'IDENTIFICATION DE LA BOUCLE SI ELLE N'EXISTE PAS          *
 *****************************************************************************/

*code_retour = OK;
*code_erreur = OK;

existence_boucle (identifiant_boucle, &cr, &ce);

if (cr == PRESENT)
   *code_erreur = ERREUR_BOUCLE;
   else {
        enregistrement_boucle (identifiant_boucle, &cr, &ce);
        if (ce == ERREUR_SYSTEME)
           *code_erreur = ENREG_BOUCLE;
        }
/*****************************************************************************
 *   RETROUVER LE CRITERE DE LA BOUCLE                                       *
 *****************************************************************************/
```

```
if (*code_erreur == OK)
   {
   critere_boucle (identifiant_boucle, &info_critere, &cr, &ce);

   if (cr == TROISIEME_CRITERE)
      info_critere.type_critere = CR_TOUS;

   if (info_critere.type_critere == CR_CONSTITUANT)
      if (strcmp (info_critere.information.critere_constituant.variable,
                  "") != 0)
         {
         lecture_variable (info_critere.information.critere_constituant.
                           variable,&valeur_variable, &cr, &ce);

         if (ce == VAR_EXISTE_PAS)
            *code_erreur = VAR_EXISTE_PAS;
            else {
                 strcpy (info_critere.information.critere_constituant.classe,
                         valeur_variable.classe);
                 strcpy (info_critere.information.critere_constituant.type,
                         valeur_variable.type);
                 strcpy (info_critere.information.critere_constituant.nom,
                         valeur_variable.nom);

                 }
         }
   }

/*****************************************************************************
 *   RECHERCHER LES OI QUI SERONT TRAITES PAR LA BOUCLE                      *
 *****************************************************************************/

if (*code_erreur == OK)
   switch (info_critere.type_critere)
          {
          case CR_CONSTITUANT :
                   strcpy (type_oi,
                     info_critere.information.critere_constituant.desir);
                   strcpy (identifiant_ois.classe,
                     info_critere.information.critere_constituant.classe);
                   strcpy (identifiant_ois.type,
                     info_critere.information.critere_constituant.type);
                   strcpy (identifiant_ois.nom,
                     info_critere.information.critere_constituant.nom);
                   identifiant_ois.num_copie = 0;

                   constituant_ois (&identifiant_ois,type_oi,liste_oi,
                                    &cr, &ce);

                   if (ce == EXISTE_PAS)
                      *code_erreur = OIS_BOUCLE;
                   break;

          case CR_CLASSE :
                   objets_classe (info_critere.information.critere_classe.
                                  classe, liste_oi, &cr, &ce);
```

```
                    break;

            case CR_TOUS :
                    tous_objets (liste_oi, &cr, &ce);
                    break;
            }

/*****************************************************************************
 *   ENREGISTRER UN OI DANS LA VARIABLE DE LA BOUCLE SI LISTE NON VIDE      *
 *****************************************************************************/

if (*code_erreur == OK)
   if (cr == VIDE)
      *code_retour = FIN_BOUCLE;
      else {
           strcpy (valeur_variable.classe, liste_oi[0]->classe);
           strcpy (valeur_variable.type, liste_oi[0]->type);
           strcpy (valeur_variable.nom, liste_oi[0]->nom);
           valeur_variable.num_copie = liste_oi[0]->num_copie;
           ptr = liste_oi[0];
           liste_oi[0] = liste_oi[0]->ptr_suivant;
           free (ptr);

           assignation_variable (info_critere.nom_variable,&valeur_variable,
                                 &cr, &ce);

           if (ce == ERREUR_SYSTEME)
              *code_erreur = ENREG_VARIABLE;
           }

/*****************************************************************************
 *   MEMORISATION DES OI SE RAPPORTANT A LA BOUCLE                           *
 *****************************************************************************/

while (*code_erreur == OK AND liste_oi[0] != NULL)
      {
      strcpy (identifiant_oi.classe, liste_oi[0]->classe);
      strcpy (identifiant_oi.type, liste_oi[0]->type);
      strcpy (identifiant_oi.nom, liste_oi[0]->nom);
      identifiant_oi.num_copie = liste_oi[0]->num_copie;

      ajouter_occurrence_boucle (&identifiant_oi, &cr, &ce);

      if (ce == ERREUR_SYSTEME)
         *code_erreur = ERREUR_SYSTEME;
         else {
              ptr = liste_oi[0];
              liste_oi[0] = liste_oi[0]->ptr_suivant;
              free (ptr);
              }
      }

#if DEBUG_BOUCLE
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<BOUCLE_FIRST.\n");
```

```
#endif
}

/*****************************************************************************/

boucle_next (identifiant_boucle,code_retour,code_erreur)

char           identifiant_boucle [MX_ID_BOUCLE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
char              var_boucle [MX_LG_VARIABLE + 1];
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_BOUCLE
  printf(">>>BOUCLE_NEXT.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 *   ENREGISTRER UN OI DANS LA VARIABLE DE LA BOUCLE SI LISTE NON VIDE      *
 *****************************************************************************/

lire_occurrence_boucle (identifiant_boucle, &identifiant_oi, &cr, &ce);

if (ce == OK)
   if (cr == VIDE)
      *code_retour = FIN_BOUCLE;
      else {
           variable_boucle (identifiant_boucle, var_boucle, &cr, &ce);

           assignation_variable (var_boucle, &identifiant_oi, &cr, &ce);

           if (ce == ERREUR_SYSTEME)
              *code_erreur = LECTURE_OBJET;
           }
   else *code_erreur = SUPPR_OBJET;

/*****************************************************************************
 *   SI LISTE VIDE ALORS SUPPRESSION DE LA BOUCLE                            *
 *****************************************************************************/

if (*code_erreur == OK AND *code_retour == FIN_BOUCLE)
   {
   suppression_boucle (identifiant_boucle, &cr, &ce);

   if (ce == ERREUR_SYSTEME)
      *code_erreur = SUPPR_BOUCLE;
   }

#if DEBUG_BOUCLE
  printf("-----------------------------------------------------\n");
```

```
  printf("Resultats :\n");
  printf("Identifiant_boucle : %s.\n",identifiant_boucle);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<BOUCLE_NEXT.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : condition.c                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des conditions.    */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

condition (identifiant_condition,code_retour,code_erreur)

char           identifiant_condition [MX_ID_CONDITION + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_clause_condition info_clause;
   ty_code_retour      code_retour_tmp;
   ty_code_erreur      code_erreur_tmp;

#if DEBUG_CONDITION
  printf(">>>CONDITION.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant condition : %s\n",identifiant_condition);
  printf("-------------------------------------------------------\n");
#endif


/*****************************************************************************
 *   RECHERCHE DE LA CLAUSE DE LA CONDITION                                  *
 *****************************************************************************/

   clause_condition (identifiant_condition,&info_clause,&code_retour_tmp,
                                                        &code_erreur_tmp);

/*****************************************************************************
 *   EVALUATION DE LA CONDITION                                              *
 *****************************************************************************/

   switch (info_clause.type_clause)
   {  case CL_ATTRIBUT : cond_attribut (&info_clause,&code_retour_tmp,
                                                     &code_erreur_tmp);
                         break;
      case CL_APPARTENANCE : cond_appartenance (&info_clause,&code_retour_tmp,
                                                            &code_erreur_tmp);
                             break;
      case CL_PARAMETRE : cond_parametre (&info_clause,&code_retour_tmp,
                                                      &code_erreur_tmp);
                          break;
   }
```

```
   *code_erreur = code_erreur_tmp;
   *code_retour = code_retour_tmp;

#if DEBUG_CONDITION
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONDITION.\n");
#endif
}

/*****************************************************************************/

cond_attribut (info_clause,code_retour,code_erreur)

st_clause_condition *info_clause;
ty_code_retour      *code_retour;
ty_code_erreur      *code_erreur;

{ char              nom_variable [MX_LG_VARIABLE + 1];
  char              nom_attribut [MX_LG_ATTRIBUT + 1];
  char              operateur [MX_LG_OPERATEUR + 1];
  char              cst_alpha_num [MX_LG_CST + 1];
  int               cst_num;
  char              valeur_attribut [MX_LG_CST + 1];
  int               valeur_attribut_num;
  int               resultat;
  st_identifiant_oi identifiant_oi;
  st_valeur_oi      info_oi;
  ty_code_retour    code_retour_tmp;
  ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONDITION
  printf(">>>COND_ATTRIBUT.\n");
#endif

   *code_retour = FAUX;
   *code_erreur = OK;

   strcpy(nom_variable,info_clause->information.clause_attribut.nom_variable);
   strcpy(nom_attribut,info_clause->information.clause_attribut.nom_attribut);
   strcpy(operateur,info_clause->information.clause_attribut.operateur);
   strcpy(cst_alpha_num,
                    info_clause->information.clause_attribut.cst_alpha_num);

/*****************************************************************************
 *   RECHERCHE DE LA VARIABLE SI NECESSAIRE                                  *
 *****************************************************************************/

   if (nom_variable[0] != '\0')
   {  lecture_variable (nom_variable,&identifiant_oi,&code_retour_tmp,
                                                     &code_erreur_tmp);
      *code_erreur = code_erreur_tmp;
   }
   else
   {  strcpy (identifiant_oi.classe,
                          info_clause->information.clause_attribut.classe);
```

```
      strcpy (identifiant_oi.type,
                        info_clause->information.clause_attribut.type);
      strcpy (identifiant_oi.nom,
                        info_clause->information.clause_attribut.nom);
      identifiant_oi.num_copie =
                        info_clause->information.clause_attribut.num_copie;
   }

/*****************************************************************************
 *   VERIFICATION DE L'EXISTENCE DE L'OI                                     *
 *****************************************************************************/

   if (*code_erreur == OK)
   {  lecture_oi (&identifiant_oi,&info_oi,&code_retour_tmp,&code_erreur_tmp);
      if (code_retour_tmp == EXISTE_PAS)
      {  if ((strcmp(identifiant_oi.classe,"MESSAGE") == 0) OR
             (strcmp(identifiant_oi.classe,"FORMULAIRE") == 0) OR
             (strcmp(identifiant_oi.classe,"DOCUMENT") == 0))

         {  *code_erreur = OIE_CONDITION;
         }
         else
         {  *code_erreur = OIS_CONDITION;
                                                  }
      }
   }

/*****************************************************************************
 *   VERIFICATION DE LA VALIDITE DU TEST ETANT DONNE LA CLASSE DE L'OI       *
 *****************************************************************************/

   if (*code_erreur ==OK)
   {  if ((((strcmp(identifiant_oi.classe,"MESSAGE") == 0) OR
            (strcmp(identifiant_oi.classe,"FORMULAIRE") == 0) OR
            (strcmp(identifiant_oi.classe,"DOCUMENT") == 0)) AND
           ((strcmp(nom_attribut,"ORDRE_CLAS") == 0) OR
            (strcmp(nom_attribut,"ETAT_CONTENU") == 0)))

       OR (((strcmp(identifiant_oi.classe,"DOSSIER") == 0) OR
            (strcmp(identifiant_oi.classe,"FICHIER") == 0) OR
            (strcmp(identifiant_oi.classe,"PILE") == 0)) AND
           ((strcmp(nom_attribut,"NUM_COPIE") == 0))))
      {  *code_erreur = CLAUSE_ATTRIBUT;
      }
   }

/*****************************************************************************
 *   EN FONCTION DE L'ATTRIBUT A TESTER ...                                  *
 *****************************************************************************/

   if (*code_erreur == OK)
   {  if (strcmp(nom_attribut,"CLASSE") == 0)
      {  strcpy (valeur_attribut,info_oi.classe);
      }
      else if (strcmp(nom_attribut,"TYPE") == 0)
      {  strcpy (valeur_attribut,info_oi.type);
      }
```

```
      else if (strcmp(nom_attribut,"NOM") == 0)
      {  strcpy (valeur_attribut,info_oi.nom);
      }
      else if (strcmp(nom_attribut,"NUM_COPIE") == 0)
      {  valeur_attribut_num = info_oi.num_copie;
      }
      else if (strcmp(nom_attribut,"ETAT") == 0)
      {  strcpy (valeur_attribut,info_oi.etat);
      }
      else if (strcmp(nom_attribut,"ECHEANCE") == 0)
      {  strcpy (valeur_attribut,info_oi.echeance);
      }
      else if (strcmp(nom_attribut,"ORDRE_CLAS") == 0)
      {  strcpy (valeur_attribut,info_oi.ordre_clas);
      }
      else if (strcmp(nom_attribut,"ETAT_CONTENU") == 0)
      {  strcpy (valeur_attribut,info_oi.etat_contenu);
      }

/*****************************************************************************
 *  ... ET DE L'OPERATEUR, EVALUATION DE LA CONDITION (ATTRIBUT NUMERIQUE) *
 *****************************************************************************/

      if (strcmp(nom_attribut,"NUM_COPIE") == 0)
      {  cst_num = atoi(cst_alpha_num);
         if (strcmp (operateur,"=") == 0)
         { if (valeur_attribut_num == cst_num) *code_retour = VRAI;
         }
         else if (strcmp (operateur,">") == 0)
         { if (valeur_attribut_num > cst_num) *code_retour = VRAI;
         }
         else if (strcmp (operateur,"<") == 0)
         { if (valeur_attribut_num < cst_num) *code_retour = VRAI;
         }
         else if (strcmp (operateur,">=") == 0)
         { if (valeur_attribut_num >= cst_num) *code_retour = VRAI;
         }
         else if (strcmp (operateur,"<=") == 0)
         { if (valeur_attribut_num <= cst_num) *code_retour = VRAI;
         }
         else if (strcmp (operateur,"!=") == 0)
         { if (valeur_attribut_num != cst_num) *code_retour = VRAI;
         }
      }

/*****************************************************************************
 *   OU EVALUATION DE LA CONDITION (ATTRIBUT ALPHA-NUMERIQUE)                *
 *****************************************************************************/

      else
      {  resultat = strcmp (valeur_attribut,cst_alpha_num);
         if (strcmp (operateur,"=") == 0)
         {  if (resultat == 0) *code_retour = VRAI;
         }
         else if (strcmp (operateur,">") == 0)
         {  if (resultat > 0) *code_retour = VRAI;
         }
```

```
            else if (strcmp (operateur,"<") == 0)
            {  if (resultat < 0) *code_retour = VRAI;
            }
            else if (strcmp (operateur,">=") == 0)
            {  if (resultat >= 0) *code_retour = VRAI;
            }
            else if (strcmp (operateur,"<=") == 0)
            {  if (resultat <= 0) *code_retour = VRAI;
            }
            else if (strcmp (operateur,"!=") == 0)
            {  if (resultat != 0) *code_retour = VRAI;
            }
         }
      }

#if DEBUG_CONDITION
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<COND_ATTRIBUT.\n");
#endif
}

/*****************************************************************************/

cond_appartenance (info_clause,code_retour,code_erreur)

st_clause_condition *info_clause;
ty_code_retour      *code_retour;
ty_code_erreur      *code_erreur;

{ char              nom_variable [MX_LG_VARIABLE + 1];
  st_identifiant_oi identifiant_oi;
  st_identifiant_oi identifiant_ois;
  ty_code_retour    code_retour_tmp;
  ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONDITION
  printf(">>>COND_APPARTENANCE.\n");
#endif

   *code_retour = FAUX;
   *code_erreur = OK;

   strcpy(nom_variable,
              info_clause->information.clause_appartenance.oi_nom_variable);

/*****************************************************************************
 *   RECHERCHE DE LA PREMIERE VARIABLE SI NECESSAIRE                         *
 *****************************************************************************/

   if (nom_variable[0] != '\0')
   {  lecture_variable (nom_variable,&identifiant_oi,&code_retour_tmp,
                                                     &code_erreur_tmp);
      *code_erreur = code_erreur_tmp;
   }
   else
```

```
   {  strcpy (identifiant_oi.classe,
                  info_clause->information.clause_appartenance.oi_classe);
      strcpy (identifiant_oi.type,
                  info_clause->information.clause_appartenance.oi_type);
      strcpy (identifiant_oi.nom,
                  info_clause->information.clause_appartenance.oi_nom);
      identifiant_oi.num_copie =
                  info_clause->information.clause_appartenance.oi_num_copie;
   }

/*****************************************************************************
 *   VERIFICATION DE LA VALIDITE DU TEST                                     *
 *****************************************************************************/

   if (*code_erreur == OK)
   {  if (strcmp(identifiant_oi.classe,"PILE") == 0)
      {  *code_erreur = CLAUSE_APPARTENANCE;
      }
   }

/*****************************************************************************
 *   RECHERCHE DE LA DEUXIEME VARIABLE SI NECESSAIRE                         *
 *****************************************************************************/

   if (*code_erreur == OK)
   {  strcpy(nom_variable,
             info_clause->information.clause_appartenance.ois_nom_variable);

      if (nom_variable[0] != '\0')
      {  lecture_variable (nom_variable,&identifiant_ois,&code_retour_tmp,
                                                    &code_erreur_tmp);
         *code_erreur = code_erreur_tmp;
      }
      else
      {  strcpy (identifiant_oi.classe,
                  info_clause->information.clause_appartenance.oi_classe);
         strcpy (identifiant_oi.type,
                  info_clause->information.clause_appartenance.oi_type);
         strcpy (identifiant_oi.nom,
                  info_clause->information.clause_appartenance.oi_nom);
         identifiant_oi.num_copie = 0;
      }
   }

/*****************************************************************************
 *   VERIFICATION DE L'EXISTENCE DES OI ET EVALUATION DE LA CONDITION        *
 *****************************************************************************/

   if (*code_erreur == OK)
   {  classement_oi (&identifiant_oi,&identifiant_ois,&code_retour_tmp,
                                                    &code_erreur_tmp);
      switch (code_erreur_tmp)

      { case OI_EXISTE_PAS :
                  if ((strcmp(identifiant_oi.classe,"MESSAGE") == 0) OR
                      (strcmp(identifiant_oi.classe,"FORMULAIRE") == 0) OR
                      (strcmp(identifiant_oi.classe,"DOCUMENT") == 0))
```

```
                        {  *code_erreur = OIE_CONDITION;
                        }
                        else
                        {  *code_erreur = OIS_CONDITION;
                                                                }
                        break;

        case OIS_EXISTE_PAS :
                        if ((strcmp(identifiant_ois.classe,"MESSAGE") == 0) OR
                            (strcmp(identifiant_ois.classe,"FORMULAIRE") == 0) OR
                            (strcmp(identifiant_ois.classe,"DOCUMENT") == 0))

                        {  *code_erreur = OIE_CONDITION;
                        }
                        else
                        {  *code_erreur = OIS_CONDITION;
                                                                }
                        break;

        case OK :       if (code_retour_tmp == CLASSEMENT)
                        { *code_retour = VRAI;
                        }
                        break;
      }
   }

#if DEBUG_CONDITION
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<COND_APPARTENANCE.\n");
#endif
}

/*****************************************************************************/

cond_parametre (info_clause,code_retour,code_erreur)

st_clause_condition *info_clause;
ty_code_retour      *code_retour;
ty_code_erreur      *code_erreur;

{  char           nom_parametre [MX_LG_PARAMETRE + 1];
   char           valeur_parametre [MX_LG_VALEUR_PARAMETRE + 1];
   char           cst_alpha_num [MX_LG_CST + 1];
   char           operateur [MX_LG_OPERATEUR + 1];
   int            resultat;
   ty_code_retour code_retour_tmp;
   ty_code_erreur code_erreur_tmp;

#if DEBUG_CONDITION
  printf(">>>COND_PARAMETRE.\n");
#endif

   *code_retour = FAUX;
   *code_erreur = OK;
```

```
   strcpy (nom_parametre,
                  info_clause->information.clause_parametre.nom_parametre);
   strcpy (cst_alpha_num,
                  info_clause->information.clause_parametre.cst_alpha_num);
   strcpy (operateur,
                  info_clause->information.clause_parametre.operateur);

/*****************************************************************************
 *   RECHERCHE DU PARAMETRE                                                  *
 *****************************************************************************/

   lecture_parametre (nom_parametre,valeur_parametre,&code_retour_tmp,
                                                     &code_erreur_tmp);

/*****************************************************************************
 *  EN FONCTION DE SA VALEUR ET DE L'OPERATEUR, EVALUATION DE LA CONDITION *
 *****************************************************************************/

   if (code_erreur_tmp != PAR_EXISTE_PAS)
   {  resultat = strcmp (valeur_parametre,cst_alpha_num);
      if (strcmp (operateur,"=") == 0)
      {  if (resultat == 0) *code_retour = VRAI;
      }
      else if (strcmp (operateur,">") == 0)
      {  if (resultat > 0) *code_retour = VRAI;
      }
      else if (strcmp (operateur,"<") == 0)
      {  if (resultat < 0) *code_retour = VRAI;
      }
      else if (strcmp (operateur,">=") == 0)
      {  if (resultat >= 0) *code_retour = VRAI;
      }
      else if (strcmp (operateur,"<=") == 0)
      {  if (resultat <= 0) *code_retour = VRAI;
      }
      else if (strcmp (operateur,"!=") == 0)
      {  if (resultat != 0) *code_retour = VRAI;
      }
   }
   else
   {  *code_erreur = PAR_EXISTE_PAS;
   }

#if DEBUG_CONDITION
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<COND_PARAMETRE.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : op_primitive.c                                                  */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des operations     */
/*               primitives.                                                 */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

operation_primitive (identifiant_operation,information_operation,code_retour,
                     code_erreur)

char           identifiant_operation [MX_ID_OPERATION + 1];
st_operation   *information_operation;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  char             type_operation [MX_TY_OPERATION + 1];
   st_operation_var info_operation_var;
   ty_code_erreur   code_erreur_tmp;
   ty_code_retour   code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OPERATION_PRIMITIVE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("identifiant operation : %s\n",identifiant_operation);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;

   sous_type_operation_primitive (identifiant_operation,&info_operation_var,
                                  &code_retour_tmp,&code_erreur_tmp);

   strcpy (type_operation,info_operation_var.type_operation);

   if (strcmp (type_operation,"CREER") == 0)
   {  op_primitive_creer (&info_operation_var,information_operation,
                          &code_erreur_tmp);
   }
   else if (strcmp (type_operation,"REPRODUIRE") == 0)
   {  op_primitive_reproduire (&info_operation_var,information_operation,
                               &code_erreur_tmp);
   }
   else if (strcmp (type_operation,"RECEVOIR") == 0)
   {  op_primitive_recevoir (&info_operation_var,information_operation,
                             &code_erreur_tmp);
   }
```

```
else if (strcmp (type_operation,"COMMUNIQUER") == 0)
{  op_primitive_communiquer (&info_operation_var,information_operation,
                             &code_erreur_tmp);
}
else if (strcmp (type_operation,"CONSULTER") == 0)
{  op_primitive_consulter (&info_operation_var,information_operation,
                           &code_erreur_tmp);
}
else if (strcmp (type_operation,"FEUILLETER") == 0)
{  op_primitive_feuilleter (&info_operation_var,information_operation,
                            &code_erreur_tmp);
}
else if (strcmp (type_operation,"ENLEVER") == 0)
{  op_primitive_enlever (&info_operation_var,information_operation,
                         &code_erreur_tmp);
}
else if (strcmp (type_operation,"MODIFIER") == 0)
{  op_primitive_modifier (&info_operation_var,information_operation,
                          &code_erreur_tmp);
}
else if (strcmp (type_operation,"COMPLETER") == 0)
{  op_primitive_completer (&info_operation_var,information_operation,
                           &code_erreur_tmp);
}
else if (strcmp (type_operation,"VERIFIER") == 0)
{  op_primitive_verifier (&info_operation_var,information_operation,
                          &code_erreur_tmp);
}
else if (strcmp (type_operation,"DETRUIRE") == 0)
{  op_primitive_detruire (&info_operation_var,information_operation,
                          &code_erreur_tmp);
}
else if (strcmp (type_operation,"ARCHIVER") == 0)
{  op_primitive_archiver (&info_operation_var,information_operation,
                          &code_erreur_tmp);
}
else if (strcmp (type_operation,"TRIER") == 0)
{  op_primitive_trier (&info_operation_var,information_operation,
                       &code_erreur_tmp);
}
else if (strcmp (type_operation,"CLASSER") == 0)
{  op_primitive_classer (&info_operation_var,information_operation,
                         &code_erreur_tmp);
}
else if (strcmp (type_operation,"RANGER") == 0)
{  op_primitive_ranger (&info_operation_var,information_operation,
                        &code_erreur_tmp);
}
else if (strcmp (type_operation,"REPLACER") == 0)
{  op_primitive_replacer (&info_operation_var,information_operation,
                          &code_erreur_tmp);
}
else if (strcmp (type_operation,"DECIDER") == 0)
{  op_primitive_decider (&info_operation_var,information_operation,
                         &code_erreur_tmp);
}
```

```
   *code_erreur = code_erreur_tmp;

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OPERATION_PRIMITIVE.\n");
#endif
}

/*****************************************************************************/

op_primitive_creer (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_creer *ptr;
   st_creer *ptr_var;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_CREER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.creer;
   ptr_var = &info_var->information.creer;

   strcpy (ptr->classe,    ptr_var->classe);
   strcpy (ptr->type  ,    ptr_var->type);
   strcpy (ptr->nom   ,    ptr_var->nom);
   strcpy (ptr->etat,      ptr_var->etat);
   strcpy (ptr->echeance,  ptr_var->echeance);
   strcpy (ptr->ordre_clas,ptr_var->ordre_clas);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_CREER.\n");
#endif
}

/*****************************************************************************/

op_primitive_reproduire (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;
```

```
{  st_reproduire     *ptr;
   st_reproduire_var *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_REPRODUIRE.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.reproduire;
   ptr_var = &info_var->information.reproduire;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   ptr->nbr_copie    = ptr_var->nbr_copie;
   ptr->num_depart   = ptr_var->num_depart;

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_REPRODUIRE.\n");
#endif
}

/*****************************************************************************/

op_primitive_recevoir (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;
```

```
{  st_recevoir *ptr;
   st_recevoir *ptr_var;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_RECEVOIR.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.recevoir;
   ptr_var = &info_var->information.recevoir;

   strcpy (ptr->classe,    ptr_var->classe);
   strcpy (ptr->type  ,    ptr_var->type);
   strcpy (ptr->nom   ,    ptr_var->nom);
   ptr->num_copie    =     ptr_var->num_copie;
   strcpy (ptr->etat,      ptr_var->etat);
   strcpy (ptr->echeance,  ptr_var->echeance);
   strcpy (ptr->ordre_clas,ptr_var->ordre_clas);
   strcpy (ptr->mode_reception,ptr_var->mode_reception);

   ptr->composants = ptr_var->composants;
   ptr_var->composants = NULL;

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_RECEVOIR.\n");
#endif
}

/*************************************************************************/

op_primitive_communiquer (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_communiquer     *ptr;
   st_communiquer_var *ptr_var;
   st_identifiant_oi  valeur_variable;
   ty_code_erreur     code_erreur_tmp;
   ty_code_retour     code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_COMMUNIQUER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
```

```
    strcpy (info->nom_operation ,info_var->nom_operation);
    strcpy (info->description   ,info_var->description);

    ptr     = &info->information.communiquer;
    ptr_var = &info_var->information.communiquer;

    if (ptr_var->nom_variable[0] != '\0')
    {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                         &code_retour_tmp,&code_erreur_tmp);
       if (code_erreur_tmp == OK)
       {  strcpy (ptr->classe,valeur_variable.classe);
          strcpy (ptr->type  ,valeur_variable.type);
          strcpy (ptr->nom   ,valeur_variable.nom);
          ptr->num_copie    = valeur_variable.num_copie;
       }
       else
       {  *code_erreur = VAR_EXISTE_PAS;
       }
    }
    else
    {  strcpy (ptr->classe,ptr_var->classe);
       strcpy (ptr->type  ,ptr_var->type);
       strcpy (ptr->nom   ,ptr_var->nom);
       ptr->num_copie    = ptr_var->num_copie;
    }
    strcpy (ptr->etat,    ptr_var->etat);
    strcpy (ptr->echeance,ptr_var->echeance);
    strcpy (ptr->mode_communication,ptr_var->mode_communication);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_COMMUNIQUER.\n");
#endif
}

/*****************************************************************************/

op_primitive_consulter (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_consulter      *ptr;
   st_consulter_var  *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_CONSULTER.\n");
#endif

   *code_erreur = OK;
```

```
   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.consulter;
   ptr_var = &info_var->information.consulter;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie     = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie     = ptr_var->num_copie;
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_CONSULTER.\n");
#endif
}

/*****************************************************************************/

op_primitive_feuilleter (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_feuilleter     *ptr;
   st_feuilleter_var *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_FEUILLETER.\n");
#endif

   *code_erreur = OK;
```

```
   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.feuilleter;
   ptr_var = &info_var->information.feuilleter;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);
   strcpy (ptr->ordre_feuilletage,ptr_var->ordre_feuilletage);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_FEUILLETER.\n");
#endif
}

/*****************************************************************************/

op_primitive_enlever (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_enlever        *ptr;
   st_enlever_var    *ptr_var;
   st_identifiant_oi valeur_variable;
   st_identifiant_oi identifiant_oi;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_ENLEVER.\n");
#endif

   *code_erreur = OK;
```

```
    strcpy (info->type_operation,info_var->type_operation);
    strcpy (info->nom_operation ,info_var->nom_operation);
    strcpy (info->description   ,info_var->description);

    ptr     = &info->information.enlever;
    ptr_var = &info_var->information.enlever;

    if (ptr_var->nom_variable[0] != '\0')
    {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                         &code_retour_tmp,&code_erreur_tmp);
       if (code_erreur_tmp == OK)
       {  if (strcmp(valeur_variable.classe,"PILE") == 0)
          {  strcpy (ptr->nom_pile,valeur_variable.nom);
          }
          else
          {  *code_erreur = CLASSE_ENLEVER;
          }
       }
       else
       {  *code_erreur = VAR_EXISTE_PAS;
       }
    }
    else
    {  strcpy (ptr->nom_pile,ptr_var->nom_pile);
    }
    if (*code_erreur == OK)
    {  premier_oi_pile (ptr->nom_pile,&identifiant_oi,&code_retour_tmp,
                                                      &code_erreur_tmp);
       if (code_erreur_tmp == OK)
       {  strcpy (ptr->classe,identifiant_oi.classe);
          strcpy (ptr->type,  identifiant_oi.type);
          strcpy (ptr->nom,   identifiant_oi.nom);
          ptr->num_copie =    identifiant_oi.num_copie;
       }
       else
       {  *code_erreur = OI_ENLEVER;
       }
    }
    if (*code_erreur == OK)
    {  assignation_variable (ptr_var->nom_variable,&identifiant_oi,
                             &code_retour_tmp,&code_erreur_tmp);
       if (code_erreur_tmp == ERREUR_SYSTEME)
       {  *code_erreur = ENREG_VARIABLE;
       }
    }

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_ENLEVER.\n");
#endif
}

/*****************************************************************************/

op_primitive_modifier (info_var,info,code_erreur)
```

```
st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_modifier       *ptr;
   st_modifier_var   *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_MODIFIER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.modifier;
   ptr_var = &info_var->information.modifier;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  if (strcmp(valeur_variable.classe,"DOCUMENT") == 0)
         {  strcpy (ptr->type  ,valeur_variable.type);
            strcpy (ptr->nom   ,valeur_variable.nom);
            ptr->num_copie    = valeur_variable.num_copie;
         }
         else
         {  *code_erreur = CLASSE_MODIFIER;
         }
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);
   strcpy (ptr->nom_message,ptr_var->nom_message);
   ptr->num_copie_message = ptr_var->num_copie_message;

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_MODIFIER.\n");
#endif
```

```
}

/*****************************************************************************/

op_primitive_completer (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_completer      *ptr;
   st_completer_var  *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_COMPLETER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.completer;
   ptr_var = &info_var->information.completer;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  if (strcmp(valeur_variable.classe,"FORMULAIRE") == 0)
         {  strcpy (ptr->type  ,valeur_variable.type);
            strcpy (ptr->nom   ,valeur_variable.nom);
            ptr->num_copie    = valeur_variable.num_copie;
         }
         else
         {  *code_erreur = CLASSE_COMPLETER;
         }
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
```

```
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_COMPLETER.\n");
#endif
}

/*****************************************************************************/

op_primitive_verifier (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_verifier       *ptr;
   st_verifier_var   *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_VERIFIER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.verifier;
   ptr_var = &info_var->information.verifier;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);
   strcpy (ptr->nom_parametre,ptr_var->nom_parametre);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
```

```
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_VERIFIER.\n");
#endif
}

/*****************************************************************************/

op_primitive_detruire (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_detruire       *ptr;
   st_detruire_var   *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_DETRUIRE.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.detruire;
   ptr_var = &info_var->information.detruire;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
```

```
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_DETRUIRE.\n");
#endif
}

/*****************************************************************************/

op_primitive_archiver (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_archiver       *ptr;
   st_archiver_var   *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_ARCHIVER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.archiver;
   ptr_var = &info_var->information.archiver;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->etat,    ptr_var->etat);
   strcpy (ptr->echeance,ptr_var->echeance);

#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
```

```
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_ARCHIVER.\n");
#endif
}

/*****************************************************************************/

op_primitive_trier (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_trier          *ptr;
   st_trier_var      *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_TRIER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.trier;
   ptr_var = &info_var->information.trier;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->etat,     ptr_var->etat);
   strcpy (ptr->echeance, ptr_var->echeance);
   strcpy (ptr->ordre_tri,ptr_var->ordre_tri);

#if DEBUG_OP_PRIMITIVE
```

```
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_TRIER.\n");
#endif
}

/*****************************************************************************/

op_primitive_classer (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_classer        *ptr;
   st_classer_var    *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_CLASSER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.classer;
   ptr_var = &info_var->information.classer;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie     = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie     = ptr_var->num_copie;
   }
   strcpy (ptr->dest_classe,ptr_var->dest_classe);
   strcpy (ptr->dest_type  ,ptr_var->dest_type);
   strcpy (ptr->dest_nom   ,ptr_var->dest_nom);
```

```
#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_CLASSER.\n");
#endif
}

/*****************************************************************************/

op_primitive_ranger (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_ranger          *ptr;
   st_ranger_var      *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur     code_erreur_tmp;
   ty_code_retour     code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_RANGER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.ranger;
   ptr_var = &info_var->information.ranger;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->dest_classe,ptr_var->dest_classe);
   strcpy (ptr->dest_nom   ,ptr_var->dest_nom);
```

```
#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_RANGER.\n");
#endif
}

/*****************************************************************************/

op_primitive_replacer (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_replacer       *ptr;
   st_replacer_var   *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_REPLACER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.replacer;
   ptr_var = &info_var->information.replacer;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->gr_classe,ptr_var->gr_classe);
   strcpy (ptr->gr_type  ,ptr_var->gr_type);
```

```
#if DEBUG_OP_PRIMITIVE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_REPLACER.\n");
#endif
}

/*****************************************************************************/

op_primitive_decider (info_var,info,code_erreur)

st_operation_var *info_var;
st_operation     *info;
ty_code_erreur   *code_erreur;

{  st_decider        *ptr;
   st_decider_var    *ptr_var;
   st_identifiant_oi valeur_variable;
   ty_code_erreur    code_erreur_tmp;
   ty_code_retour    code_retour_tmp;

#if DEBUG_OP_PRIMITIVE
  printf(">>>OP_PRIMITIVE_DECIDER.\n");
#endif

   *code_erreur = OK;

   strcpy (info->type_operation,info_var->type_operation);
   strcpy (info->nom_operation ,info_var->nom_operation);
   strcpy (info->description   ,info_var->description);

   ptr     = &info->information.decider;
   ptr_var = &info_var->information.decider;

   if (ptr_var->nom_variable[0] != '\0')
   {  lecture_variable (ptr_var->nom_variable,&valeur_variable,
                        &code_retour_tmp,&code_erreur_tmp);
      if (code_erreur_tmp == OK)
      {  strcpy (ptr->classe,valeur_variable.classe);
         strcpy (ptr->type  ,valeur_variable.type);
         strcpy (ptr->nom   ,valeur_variable.nom);
         ptr->num_copie    = valeur_variable.num_copie;
      }
      else
      {  *code_erreur = VAR_EXISTE_PAS;
      }
   }
   else
   {  strcpy (ptr->classe,ptr_var->classe);
      strcpy (ptr->type  ,ptr_var->type);
      strcpy (ptr->nom   ,ptr_var->nom);
      ptr->num_copie    = ptr_var->num_copie;
   }
   strcpy (ptr->nom_parametre,ptr_var->nom_parametre);

#if DEBUG_OP_PRIMITIVE
```

```
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OP_PRIMITIVE_DECIDER.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : variable.h                                                      */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient les declarations du module de gestion */
/*               des variables.                                              */
/*                                                                           */
/*****************************************************************************/

typedef struct lst_variables
              { char                 nom_variable [MX_LG_VARIABLE + 1];
                st_identifiant_oi    valeur_variable;
                struct lst_variables *ptr_suivant;
              }
        st_liste_variables;

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : variable.c                                                      */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des variables.     */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"
#include "variable.h"

/*****************************************************************************/

static st_liste_variables *liste_variables;

/*****************************************************************************/

initialisation_variable ()

{
#if DEBUG_VARIABLE
  printf(">>>INITIALISATION_VARIABLE.\n");
#endif

   liste_variables = NULL;

#if DEBUG_VARIABLE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des variables : \n");
  printf("-------------------------------------------------------\n");
  printf("<<<INITIALISATION_VARIABLE.\n");
#endif
}

/*****************************************************************************/

assignation_variable (nom_variable,valeur_variable,code_retour,code_erreur)

char              nom_variable [MX_LG_VARIABLE + 1];
st_identifiant_oi *valeur_variable;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  BOOLEAN            trouve;
   st_liste_variables *ptr_courant;

#if DEBUG_VARIABLE
  printf(">>>ASSIGNATION_VARIABLE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom_variable    : %s.\n",nom_variable);
  printf("Valeur_variable : \n");
  printf("Classe    : %s.\n",valeur_variable->classe);
  printf("Type      : %s.\n",valeur_variable->type);
```

```
  printf("Nom       : %s.\n",valeur_variable->nom);
  printf("Num_copie : %d.\n",valeur_variable->num_copie);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;
   trouve       = FALSE;
   ptr_courant  = liste_variables;

   while ((trouve == FALSE) AND (ptr_courant != NULL))
   {  if (strcmp (ptr_courant->nom_variable,nom_variable) == 0)
      {  trouve = TRUE;
         strcpy (ptr_courant->valeur_variable.classe,valeur_variable->classe);
         strcpy (ptr_courant->valeur_variable.type,  valeur_variable->type);
         strcpy (ptr_courant->valeur_variable.nom,   valeur_variable->nom);
         ptr_courant->valeur_variable.num_copie = valeur_variable->num_copie;
      }
      else
      {  ptr_courant = ptr_courant->ptr_suivant;
      }
   }
   if (trouve == FALSE)
   {  ptr_courant = (st_liste_variables*) malloc(sizeof(st_liste_variables));
      if (ptr_courant != NULL)
      {  strcpy (ptr_courant->nom_variable,nom_variable);
         strcpy (ptr_courant->valeur_variable.classe,valeur_variable->classe);
         strcpy (ptr_courant->valeur_variable.type,  valeur_variable->type);
         strcpy (ptr_courant->valeur_variable.nom,   valeur_variable->nom);
         ptr_courant->valeur_variable.num_copie = valeur_variable->num_copie;
         ptr_courant->ptr_suivant = liste_variables;
         liste_variables = ptr_courant;
      }
      else
      {  *code_erreur = ERREUR_SYSTEME;
      }
   }

#if DEBUG_VARIABLE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des variables : \n");
  ptr_courant = liste_variables;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_variable : %s.\n",ptr_courant->nom_variable);
    printf("Valeur_variable :\n");
    printf("Classe    : %s.\n",ptr_courant->valeur_variable.classe);
    printf("Type      : %s.\n",ptr_courant->valeur_variable.type);
    printf("Nom       : %s.\n",ptr_courant->valeur_variable.nom);
    printf("Num_copie : %d.\n",ptr_courant->valeur_variable.num_copie);
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<ASSIGNATION_VARIABLE.\n");
#endif
```

```
}

/*****************************************************************************/

lecture_variable (nom_variable,valeur_variable,code_retour,code_erreur)

char              nom_variable [MX_LG_VARIABLE + 1];
st_identifiant_oi *valeur_variable;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  BOOLEAN           trouve;
   st_liste_variables *ptr_courant;

#if DEBUG_VARIABLE
  printf(">>>LECTURE_VARIABLE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom_variable    : %s.\n",nom_variable);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;
   trouve       = FALSE;
   ptr_courant  = liste_variables;

   while ((trouve == FALSE) AND (ptr_courant != NULL))
   {  if (strcmp (ptr_courant->nom_variable,nom_variable) == 0)
      {  trouve = TRUE;
         strcpy (valeur_variable->classe,ptr_courant->valeur_variable.classe);
         strcpy (valeur_variable->type,  ptr_courant->valeur_variable.type);
         strcpy (valeur_variable->nom,   ptr_courant->valeur_variable.nom);
         valeur_variable->num_copie = ptr_courant->valeur_variable.num_copie;
      }
      else
      {  ptr_courant = ptr_courant->ptr_suivant;
      }
   }
   if (trouve == FALSE)
   {  *code_erreur = VAR_EXISTE_PAS;
   }

#if DEBUG_VARIABLE
  printf("-------------------------------------------------------\n");
  printf("Valeur_variable : \n");
  printf("Classe    : %s.\n",valeur_variable->classe);
  printf("Type      : %s.\n",valeur_variable->type);
  printf("Nom       : %s.\n",valeur_variable->nom);
  printf("Num_copie : %d.\n",valeur_variable->num_copie);
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des variables : \n");
  ptr_courant = liste_variables;   while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_variable : %s.\n",ptr_courant->nom_variable);
    printf("Valeur_variable :\n");
    printf("Classe    : %s.\n",ptr_courant->valeur_variable.classe);
```

```
    printf("Type      : %s.\n",ptr_courant->valeur_variable.type);
    printf("Nom       : %s.\n",ptr_courant->valeur_variable.nom);
    printf("Num_copie : %d.\n",ptr_courant->valeur_variable.num_copie);
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<LECTURE_VARIABLE.\n");
#endif
}

/*****************************************************************************/

suppression_variable (code_erreur)

ty_code_erreur    *code_erreur;

{  st_liste_variables *ptr_courant;

#if DEBUG_VARIABLE
  printf(">>>SUPPRESSION_VARIABLE.\n");
#endif

   *code_erreur = OK;

   while (liste_variables != NULL)
   {  ptr_courant = liste_variables;
      liste_variables = liste_variables->ptr_suivant;
      free (ptr_courant);
   }

#if DEBUG_VARIABLE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des variables : \n");
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SUPPRESSION_VARIABLE.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : parametre.h                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient les declarations du module de gestion */
/*               des parametres.                                             */
/*                                                                           */
/*****************************************************************************/

typedef struct lst_parametres
              { char nom_parametre    [MX_LG_PARAMETRE + 1];
                char valeur_parametre [MX_LG_VALEUR_PARAMETRE + 1];
                struct lst_parametres *ptr_suivant;
              }
       st_liste_parametres;

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : parametre.c                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des parametres.    */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"
#include "parametre.h"

/*****************************************************************************/

static st_liste_parametres *liste_parametres;

/*****************************************************************************/

initialisation_parametre ()

{
#if DEBUG_PARAMETRE
  printf(">>>INITIALISATION_PARAMETRE.\n");
#endif

   liste_parametres = NULL;

#if DEBUG_PARAMETRE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des parametres : \n");
  printf("-------------------------------------------------------\n");
  printf("<<<INITIALISATION_PARAMETRE.\n");
#endif
}

/*****************************************************************************/

assignation_parametre (nom_parametre,valeur_parametre,code_retour,code_erreur)

char           nom_parametre    [MX_LG_PARAMETRE + 1];
char           valeur_parametre [MX_LG_VALEUR_PARAMETRE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  BOOLEAN             trouve;
   st_liste_parametres *ptr_courant;

#if DEBUG_PARAMETRE
  printf(">>>ASSIGNATION_PARAMETRE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom_parametre    : %s.\n",nom_parametre);
  printf("Valeur_parametre : %s.\n",valeur_parametre);
  printf("-------------------------------------------------------\n");
#endif
```

```
    *code_erreur = OK;
    *code_retour = OK;
    trouve       = FALSE;
    ptr_courant  = liste_parametres;

    while ((trouve == FALSE) AND (ptr_courant != NULL))
    {  if (strcmp (ptr_courant->nom_parametre,nom_parametre) == 0)
       {  trouve = TRUE;
          strcpy(ptr_courant->valeur_parametre,valeur_parametre);
       }
       else
       {  ptr_courant = ptr_courant->ptr_suivant;
       }
    }
    if (trouve == FALSE)
    {  ptr_courant = (st_liste_parametres *)
                     malloc(sizeof(st_liste_parametres));
       if (ptr_courant != NULL)
       {  strcpy (ptr_courant->nom_parametre,nom_parametre);
          strcpy (ptr_courant->valeur_parametre,valeur_parametre);
          ptr_courant->ptr_suivant = liste_parametres;
          liste_parametres = ptr_courant;
       }
       else
       {  *code_erreur = ERREUR_SYSTEME;
       }
    }

#if DEBUG_PARAMETRE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des parametres : \n");
  ptr_courant = liste_parametres;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_parametre    : %s.\n",ptr_courant->nom_parametre);
    printf("Valeur_parametre : %s.\n",ptr_courant->valeur_parametre);
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<ASSIGNATION_PARAMETRE.\n");
#endif
}

/*****************************************************************************/

lecture_parametre (nom_parametre,valeur_parametre,code_retour,code_erreur)

char           nom_parametre    [MX_LG_PARAMETRE + 1];
char           valeur_parametre [MX_LG_VALEUR_PARAMETRE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  BOOLEAN             trouve;
   st_liste_parametres *ptr_courant;
```

```
#if DEBUG_PARAMETRE
  printf(">>>LECTURE_PARAMETRE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom_parametre    : %s.\n",nom_parametre);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;
   trouve       = FALSE;
   ptr_courant  = liste_parametres;

   while ((trouve == FALSE) AND (ptr_courant != NULL))
   {  if (strcmp (ptr_courant->nom_parametre,nom_parametre) == 0)
      {  trouve = TRUE;
         strcpy(valeur_parametre,ptr_courant->valeur_parametre);
      }
      else
      {  ptr_courant = ptr_courant->ptr_suivant;
      }
   }
   if (trouve == FALSE)
   {  *code_erreur = PAR_EXISTE_PAS;
   }

#if DEBUG_PARAMETRE
  printf("-------------------------------------------------------\n");
  printf("Valeur_parametre : %s.\n",valeur_parametre);
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des parametres : \n");
  ptr_courant = liste_parametres;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_parametre    : %s.\n",ptr_courant->nom_parametre);
    printf("Valeur_parametre : %s.\n",ptr_courant->valeur_parametre);
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<LECTURE_PARAMETRE.\n");
#endif
}

/*****************************************************************************/

suppression_parametre (code_erreur)

ty_code_erreur *code_erreur;

{  st_liste_parametres *ptr_courant;

#if DEBUG_PARAMETRE
  printf(">>>SUPPRESSION_PARAMETRE.\n");
#endif
```

```
   *code_erreur = OK;

   while (liste_parametres != NULL)
   {  ptr_courant = liste_parametres;
      liste_parametres = liste_parametres->ptr_suivant;
      free (ptr_courant);
   }

#if DEBUG_PARAMETRE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des parametres : \n");
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SUPPRESSION_PARAMETRE.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : memo_schema.h                                                   */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient les declarations du module de gestion */
/*               de la memorisation des sous-schemas.                        */
/*                                                                           */
/*****************************************************************************/

typedef struct lst_ss_schemas
              { char                  nom_ss_schema [MX_ID_SOUS_SCHEMA + 1];
                struct lst_ss_schemas *ptr_suivant;
              }
       st_liste_ss_schemas;

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : memo_schema.c                                                   */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion de la memo-        */
/*               risation des sous-schemas.                                  */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"
#include "memo_schema.h"

/*****************************************************************************/

static st_liste_ss_schemas *liste_ss_schemas;

/*****************************************************************************/

initialisation_sous_schema ()

{
#if DEBUG_MEMO_SOUS_SCHEMA
  printf(">>>INITIALISATION_SOUS_SCHEMA.\n");
#endif

   liste_ss_schemas = NULL;

#if DEBUG_MEMO_SOUS_SCHEMA
  printf("-----------------------------------------------------\n");
  printf("Resultat de la liste des sous_schemas : \n");
  printf("-----------------------------------------------------\n");
  printf("<<<INITIALISATION_SOUS_SCHEMA.\n");
#endif
}

/*****************************************************************************/

enregistrement_sous_schema (identifiant_sous_schema,code_retour,code_erreur)

char           identifiant_sous_schema [MX_ID_SOUS_SCHEMA + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_liste_ss_schemas *ptr_courant;

#if DEBUG_MEMO_SOUS_SCHEMA
  printf(">>>ENREGISTREMENT_SOUS_SCHEMA.\n");
  printf("-----------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_sous_schema : %s.\n",identifiant_sous_schema);
  printf("-----------------------------------------------------\n");
#endif

   *code_erreur = OK;
```

```
   *code_retour = OK;

   ptr_courant = (st_liste_ss_schemas*) malloc(sizeof(st_liste_ss_schemas));
   if (ptr_courant != NULL)
   {  strcpy (ptr_courant->nom_ss_schema,identifiant_sous_schema);
      ptr_courant->ptr_suivant = liste_ss_schemas;
      liste_ss_schemas = ptr_courant;
   }
   else
   {  *code_erreur = ERREUR_SYSTEME;
   }

#if DEBUG_MEMO_SOUS_SCHEMA
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des sous_schemas : \n");
  ptr_courant = liste_ss_schemas;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_ss_schema : %s.\n",ptr_courant->nom_ss_schema);
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<ENREGISTREMENT_SOUS_SCHEMA.\n");
#endif
}

/*****************************************************************************/

existence_sous_schema (identifiant_sous_schema,code_retour,code_erreur)

char           identifiant_sous_schema [MX_ID_SOUS_SCHEMA + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  BOOLEAN             trouve;
   st_liste_ss_schemas *ptr_courant;

#if DEBUG_MEMO_SOUS_SCHEMA
  printf(">>>EXISTENCE_SOUS_SCHEMA.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_sous_schema : %s.\n",identifiant_sous_schema);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = ABSENT;
   trouve       = FALSE;
   ptr_courant  = liste_ss_schemas;

   while ((trouve == FALSE) AND (ptr_courant != NULL))
   {  if (strcmp (ptr_courant->nom_ss_schema,identifiant_sous_schema) == 0)
      {  trouve       = TRUE;
         *code_retour = PRESENT;
      }
```

```
      else
      {  ptr_courant = ptr_courant->ptr_suivant;
      }
   }

#if DEBUG_MEMO_SOUS_SCHEMA
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des sous_schemas : \n");
  ptr_courant = liste_ss_schemas;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_ss_schema : %s.\n",ptr_courant->nom_ss_schema);
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<EXISTENCE_SOUS_SCHEMA.\n");
#endif
}

/*************************************************************************/

suppression_sous_schema (identifiant_sous_schema,code_retour,code_erreur)

char           identifiant_sous_schema [MX_ID_SOUS_SCHEMA + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_liste_ss_schemas *ptr_courant;

#if DEBUG_MEMO_SOUS_SCHEMA
  printf(">>>SUPPRESSION_SOUS_SCHEMA.\n");
#endif

   *code_erreur     = OK;
   *code_retour     = OK;
   ptr_courant      = liste_ss_schemas;
   liste_ss_schemas = liste_ss_schemas->ptr_suivant;

   strcpy (identifiant_sous_schema,ptr_courant->nom_ss_schema);
   free (ptr_courant);
   if (liste_ss_schemas == NULL)
   {  *code_retour = DERNIER;
   }

#if DEBUG_MEMO_SOUS_SCHEMA
  printf("-------------------------------------------------------\n");
  printf("Identifiant_sous_schema : %s.\n",identifiant_sous_schema);
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des sous_schemas : \n");
  ptr_courant = liste_ss_schemas;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_ss_schema : %s.\n",ptr_courant->nom_ss_schema);
    ptr_courant = ptr_courant->ptr_suivant;
  }
```

```
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SUPPRESSION_SOUS_SCHEMA.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : memo_boucle.h                                                   */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient les declarations du module de gestion */
/*               de la memorisation des boucles.                             */
/*                                                                           */
/*****************************************************************************/

typedef struct lst_boucles
              { char                    nom_boucle [MX_ID_BOUCLE + 1];
                st_liste_identifiant_oi *composants;
                struct lst_boucles      *ptr_suivant;
              }
        st_liste_boucles;

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : memo_boucle.c                                                   */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion de la memo-        */
/*               risation des boucles.                                       */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"
#include "memo_boucle.h"

/*****************************************************************************/

static st_liste_boucles *liste_boucles;

/*****************************************************************************/

initialisation_boucle ()

{
#if DEBUG_MEMO_BOUCLE
  printf(">>>INITIALISATION_BOUCLE.\n");
#endif

   liste_boucles = NULL;

#if DEBUG_MEMO_BOUCLE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des boucles : \n");
  printf("-------------------------------------------------------\n");
  printf("<<<INITIALISATION_BOUCLE.\n");
#endif
}

/*****************************************************************************/

enregistrement_boucle (identifiant_boucle,code_retour,code_erreur)

char           identifiant_boucle [MX_ID_BOUCLE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_liste_boucles *ptr_courant;

#if DEBUG_MEMO_BOUCLE
  printf(">>>ENREGISTREMENT_BOUCLE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_boucle : %s.\n",identifiant_boucle);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
```

```
    *code_retour = OK;

    ptr_courant = (st_liste_boucles*) malloc(sizeof(st_liste_boucles));
    if (ptr_courant != NULL)
    {  strcpy (ptr_courant->nom_boucle,identifiant_boucle);
       ptr_courant->composants  = NULL;
       ptr_courant->ptr_suivant = liste_boucles;
       liste_boucles = ptr_courant;
    }
    else
    {  *code_erreur = ERREUR_SYSTEME;
    }

#if DEBUG_MEMO_BOUCLE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des boucles : \n");
  ptr_courant = liste_boucles;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_boucle : %s.\n",ptr_courant->nom_boucle);
    printf("Liste des oi associes : \n");

    { st_liste_identifiant_oi *ptr;

      ptr = ptr_courant->composants;
      while (ptr != NULL)
      { printf("\n");
        printf("Classe    : %s.\n",ptr->classe);
        printf("Type      : %s.\n",ptr->type);
        printf("Nom       : %s.\n",ptr->nom);
        printf("Num_copie : %d.\n",ptr->num_copie);
        ptr= ptr->ptr_suivant;
      }
    }
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<ENREGISTREMENT_BOUCLE.\n");
#endif
}

/*****************************************************************************/

existence_boucle (identifiant_boucle,code_retour,code_erreur)

char           identifiant_boucle [MX_ID_BOUCLE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  BOOLEAN          trouve;
   st_liste_boucles *ptr_courant;

#if DEBUG_MEMO_BOUCLE
  printf(">>>EXISTENCE_BOUCLE.\n");
  printf("-------------------------------------------------------\n");
```

```
  printf("Arguments :\n");
  printf("Identifiant_boucle : %s.\n",identifiant_boucle);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = ABSENT;
   trouve       = FALSE;
   ptr_courant  = liste_boucles;

   while ((trouve == FALSE) AND (ptr_courant != NULL))
   {  if (strcmp (ptr_courant->nom_boucle,identifiant_boucle) == 0)
      {  trouve       = TRUE;
         *code_retour = PRESENT;
      }
      else
      {  ptr_courant = ptr_courant->ptr_suivant;
      }
   }

#if DEBUG_MEMO_BOUCLE
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des boucles : \n");
  ptr_courant = liste_boucles;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_boucle : %s.\n",ptr_courant->nom_boucle);
    printf("Liste des oi associes : \n");

    { st_liste_identifiant_oi *ptr;

      ptr = ptr_courant->composants;
      while (ptr != NULL)
      { printf("\n");
        printf("Classe    : %s.\n",ptr->classe);
        printf("Type      : %s.\n",ptr->type);
        printf("Nom       : %s.\n",ptr->nom);
        printf("Num_copie : %d.\n",ptr->num_copie);
        ptr= ptr->ptr_suivant;
      }
    }
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<EXISTENCE_BOUCLE.\n");
#endif
}

/*****************************************************************************/

suppression_boucle (identifiant_boucle,code_retour,code_erreur)

char           identifiant_boucle [MX_ID_BOUCLE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;
```

```
{  st_liste_boucles *ptr_courant;

#if DEBUG_MEMO_BOUCLE
  printf(">>>SUPPRESSION_BOUCLE.\n");
#endif

   *code_erreur     = OK;
   *code_retour     = OK;
   ptr_courant      = liste_boucles;
   liste_boucles = liste_boucles->ptr_suivant;

   strcpy (identifiant_boucle,ptr_courant->nom_boucle);
   free (ptr_courant);

#if DEBUG_MEMO_BOUCLE
  printf("-------------------------------------------------------\n");
  printf("Identifiant_boucle : %s.\n",identifiant_boucle);
  printf("-------------------------------------------------------\n");
  printf("Resultat de la liste des boucles : \n");
  ptr_courant = liste_boucles;
  while (ptr_courant != NULL)
  { printf("-------------------------------------------------------\n");
    printf("Nom_boucle : %s.\n",ptr_courant->nom_boucle);
    printf("Liste des oi associes : \n");

    { st_liste_identifiant_oi *ptr;

      ptr = ptr_courant->composants;
      while (ptr != NULL)
      { printf("\n");
        printf("Classe    : %s.\n",ptr->classe);
        printf("Type      : %s.\n",ptr->type);
        printf("Nom       : %s.\n",ptr->nom);
        printf("Num_copie : %d.\n",ptr->num_copie);
        ptr= ptr->ptr_suivant;
      }
    }
    ptr_courant = ptr_courant->ptr_suivant;
  }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SUPPRESSION_BOUCLE.\n");
#endif
}

/*****************************************************************************/

ajouter_occurrence_boucle (identifiant_oi,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{ st_liste_identifiant_oi *ptr_courant;
```

```
#if DEBUG_MEMO_BOUCLE
  printf(">>>AJOUTER_OCCURRENCE_BOUCLE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_oi : \n");
  printf("Classe     : %s.\n",identifiant_oi->classe);
  printf("Type       : %s.\n",identifiant_oi->type);
  printf("Nom        : %s.\n",identifiant_oi->nom);
  printf("Num_copie  : %d.\n",identifiant_oi->num_copie);
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;

   ptr_courant = (st_liste_identifiant_oi*)
                 malloc(sizeof(st_liste_identifiant_oi));
   if (ptr_courant != NULL)
   {  strcpy (ptr_courant->classe,identifiant_oi->classe);
      strcpy (ptr_courant->type  ,identifiant_oi->type);
      strcpy (ptr_courant->nom   ,identifiant_oi->nom);
      ptr_courant->num_copie    = identifiant_oi->num_copie;
      ptr_courant->ptr_suivant  = liste_boucles->composants;
      liste_boucles->composants = ptr_courant;
   }
   else
   {  *code_erreur = ERREUR_SYSTEME;
   }
#if DEBUG_MEMO_BOUCLE
  { st_liste_boucles *ptr_liste;

    printf("-------------------------------------------------------\n");
    printf("Resultat de la liste des boucles : \n");
    ptr_liste = liste_boucles;
    while (ptr_liste != NULL)
    { printf("-------------------------------------------------------\n");
      printf("Nom_boucle : %s.\n",ptr_liste->nom_boucle);
      printf("Liste des oi associes : \n");

      { st_liste_identifiant_oi *ptr;

        ptr = ptr_liste->composants;
        while (ptr != NULL)
        { printf("\n");
          printf("Classe     : %s.\n",ptr->classe);
          printf("Type       : %s.\n",ptr->type);
          printf("Nom        : %s.\n",ptr->nom);
          printf("Num_copie  : %d.\n",ptr->num_copie);
          ptr= ptr->ptr_suivant;
        }
      }
      ptr_liste = ptr_liste->ptr_suivant;
    }
    printf("-------------------------------------------------------\n");
    printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
    printf("-------------------------------------------------------\n");
  }
```

```
  printf("<<<AJOUTER_OCCURRENCE_BOUCLE.\n");
#endif
}

/*****************************************************************************/

lire_occurrence_boucle (identifiant_boucle,identifiant_oi,code_retour,
                                                          code_erreur)

char              identifiant_boucle [MX_ID_BOUCLE + 1];
st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_liste_identifiant_oi *ptr_courant;

#if DEBUG_MEMO_BOUCLE
  printf(">>>LIRE_OCCURRENCE_BOUCLE.\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;

   strcpy (identifiant_boucle,liste_boucles->nom_boucle);
   if (liste_boucles->composants != NULL)
   {  ptr_courant               = liste_boucles->composants;
      liste_boucles->composants = ptr_courant->ptr_suivant;
      strcpy (identifiant_oi->classe,ptr_courant->classe);
      strcpy (identifiant_oi->type  ,ptr_courant->type);
      strcpy (identifiant_oi->nom   ,ptr_courant->nom);
      identifiant_oi->num_copie = ptr_courant->num_copie;
      free (ptr_courant);
   }
   else
   {  *code_retour = VIDE;
   }

#if DEBUG_MEMO_BOUCLE
  { st_liste_boucles *ptr_liste;

    printf("-----------------------------------------------------\n");
    printf("Identifiant boucle : %s.\n",identifiant_boucle);
    printf("Identifiant_oi : \n");
    printf("Classe     : %s.\n",identifiant_oi->classe);
    printf("Type       : %s.\n",identifiant_oi->type);
    printf("Nom        : %s.\n",identifiant_oi->nom);
    printf("Num_copie : %d.\n",identifiant_oi->num_copie);
    printf("-----------------------------------------------------\n");
    printf("Resultat de la liste des boucles : \n");
    ptr_liste = liste_boucles;
    while (ptr_liste != NULL)
    { printf("-----------------------------------------------------\n");
      printf("Nom_boucle : %s.\n",ptr_liste->nom_boucle);
      printf("Liste des oi associes : \n");

      { st_liste_identifiant_oi *ptr;
```

```
        ptr = ptr_liste->composants;
        while (ptr != NULL)
        { printf("\n");
          printf("Classe    : %s.\n",ptr->classe);
          printf("Type      : %s.\n",ptr->type);
          printf("Nom       : %s.\n",ptr->nom);
          printf("Num_copie : %d.\n",ptr->num_copie);
          ptr= ptr->ptr_suivant;
        }
      }
      ptr_liste = ptr_liste->ptr_suivant;
    }
    printf("-------------------------------------------------------\n");
    printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
    printf("-------------------------------------------------------\n");
  }
  printf("<<<LIRE_OCCURRENCE_BOUCLE.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : controles.c                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des controles a    */
/*               priori.                                                     */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

controles_a_priori (information_op_prim,code_retour,code_erreur)

st_operation   *information_op_prim;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  char           type_operation [MX_TY_OPERATION + 1];
   ty_code_erreur code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI.\n");
#endif

*code_erreur = OK;
*code_retour = OK;

strcpy (type_operation,information_op_prim->type_operation);

if (strcmp (type_operation,"CREER") == 0)
{  controles_a_priori_creer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"REPRODUIRE") == 0)
{  controles_a_priori_reproduire (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"RECEVOIR") == 0)
{  controles_a_priori_recevoir (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"COMMUNIQUER") == 0)
{  controles_a_priori_communiquer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"CONSULTER") == 0)
{  controles_a_priori_consulter (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"FEUILLETER") == 0)
{  controles_a_priori_feuilleter (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"ENLEVER") == 0)
{  controles_a_priori_enlever (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"MODIFIER") == 0)
{  controles_a_priori_modifier (information_op_prim,&code_erreur_tmp);
```

```
}
else if (strcmp (type_operation,"COMPLETER") == 0)
{  controles_a_priori_completer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"VERIFIER") == 0)
{  controles_a_priori_verifier (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"DETRUIRE") == 0)
{  controles_a_priori_detruire (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"ARCHIVER") == 0)
{  controles_a_priori_archiver (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"TRIER") == 0)
{  controles_a_priori_trier (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"CLASSER") == 0)
{  controles_a_priori_classer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"RANGER") == 0)
{  controles_a_priori_ranger (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"REPLACER") == 0)
{  controles_a_priori_replacer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"DECIDER") == 0)
{  controles_a_priori_decider (information_op_prim,&code_erreur_tmp);
}

if (code_erreur_tmp != OK)
{  message_erreur (code_erreur_tmp);
   *code_erreur = ERREUR;
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_creer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_creer          *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_CREER.\n");
```

```
#endif

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL N'EXISTE PAS                          *
 *****************************************************************************/

ptr_courant = &(information_op_prim->information.creer);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie = 0;

lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

if (code_retour_tmp != EXISTE_PAS)
{  *code_erreur = OI_CREER;
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_CREER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_reproduire (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
char              classe [MX_LG_CLASSE + 1];
ty_code_retour    cr;
ty_code_erreur    ce;
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_REPRODUIRE.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OBJET A REPRODUIRE                                *
 *****************************************************************************/

strcpy (classe, information_op_prim->information.reproduire.classe);

if (strcmp (classe, "MESSAGE") != 0 AND
    strcmp (classe, "FORMULAIRE") != 0 AND
    strcmp (classe, "DOCUMENT") != 0)
```

```
    *code_erreur = CLASSE_REPRODUIRE;

/*****************************************************************************
 * VERIFIER QUE LA PHOTOCOPIEUSE EXISTE                                      *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   lecture_objet ("PHOTOCOPIEUSE", "PHOTOCOPIEUSE", OUTIL_TRAVAIL, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = PHOTOCOPIEUSE_EXISTE_PAS;
   }

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL ELEMENTAIRE EXISTE                    *
 *****************************************************************************/

 if (*code_erreur == OK)
    {
    strcpy (identifiant_oi.classe,
                      information_op_prim->information.reproduire.classe);
    strcpy (identifiant_oi.type,
                      information_op_prim->information.reproduire.type);
    strcpy (identifiant_oi.nom,
                      information_op_prim->information.reproduire.nom);
    identifiant_oi.num_copie =
                      information_op_prim->information.reproduire.num_copie;

    lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

    if (cr == EXISTE_PAS)
       *code_erreur = OIE_REPRODUIRE;
    }

/*****************************************************************************
 * VERIFIER QUE LE NUMERO DE DEPART EST > QUE CEUX EXISTANT POUR CET OI       *
 *****************************************************************************/

/*****************************************************************************
 * !!!!! UNE COPIE D'UNE COPIE DEVIENT COPIE DE L'ORIGINAL S'IL EXISTE       *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   numero_copie (&identifiant_oi,&information_op_prim->information.reproduire.
                 num_depart, &cr, &ce);

   if (cr == INFERIEUR)
      *code_erreur = NUM_DEPART_INCORRECT;
   }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
```

```
  printf("<<<CONTROLES_A_PRIORI_REPRODUIRE.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_recevoir (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi  identifiant_oi;
   st_valeur_oi       valeur_oi;
   st_recevoir        *ptr_courant;
   st_liste_valeur_oi *ptr_composant;
   ty_code_retour     code_retour_tmp;
   ty_code_erreur     code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_RECEVOIR.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.recevoir);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie =   ptr_courant->num_copie;

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL N'EXISTE PAS                          *
 *****************************************************************************/

lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

if (code_retour_tmp != EXISTE_PAS)
{  *code_erreur = OI_RECEVOIR;
}

/*****************************************************************************
 * VERIFIER LES COMPOSANTS N'EXISTENT PAS                                    *

*****************************************************************************/

ptr_composant = ptr_courant->composants;
while ((ptr_composant != NULL) AND (*code_erreur == OK))
{  strcpy(identifiant_oi.classe,ptr_composant->classe);
   strcpy(identifiant_oi.type  ,ptr_composant->type);
   strcpy(identifiant_oi.nom   ,ptr_composant->nom);
   identifiant_oi.num_copie =   ptr_composant->num_copie;

   lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

   if (code_retour_tmp != EXISTE_PAS)
   {  *code_erreur = OI_RECEVOIR;
   }
```

```
   else
   {  ptr_composant = ptr_composant->ptr_suivant;
   }
}

/*****************************************************************************
 * VERIFIER QUE LE TELEPHONE EXISTE OU QUE LA BOITE IN EXISTE                *
 *****************************************************************************/

if (*code_erreur == OK)
{  if (strcmp(ptr_courant->mode_reception,"ECRIT") == 0)
   {  lecture_objet("BOITE","IN",OBJET_INTERFACE,
                                        &code_retour_tmp,&code_erreur_tmp);
      if (code_retour_tmp == EXISTE_PAS)
      {  *code_erreur = BOITE_IN_EXISTE_PAS;
      }
   }
   else
   {  lecture_objet("TELEPHONE","TELEPHONE",OBJET_INTERFACE,
                                        &code_retour_tmp,&code_erreur_tmp);
      if (code_retour_tmp == EXISTE_PAS)
      {  *code_erreur = TELEPHONE_EXISTE_PAS;
      }
   }
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_RECEVOIR.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_communiquer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_COMMUNIQUER.\n");
#endif

/*****************************************************************************
 * TEST DU MODE DE COMMUNICATION ET TRAITEMENT EN CONSEQUENCE                *
 *****************************************************************************/

if (strcmp (information_op_prim->information.communiquer.mode_communication,
            "ECRIT") == 0)
   c_com_ecrit (information_op_prim, code_erreur);
   else c_com_oral (information_op_prim, code_erreur);

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
```

```
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_COMMUNIQUER.\n");
#endif
}

/*****************************************************************************
 * TRAITEMENT D'UNE COMMUNICATION ECRITE                                     *
 *****************************************************************************/

c_com_ecrit (information_op_prim, code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;
ty_code_retour    cr;
ty_code_erreur    ce;

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER QUE L'OBJET TRANSMIS N'EST PAS UNE PILE                          *
 *****************************************************************************/

if (strcmp (information_op_prim->information.communiquer.classe, "PILE") == 0)
   *code_erreur = CLASSE_COMMUNIQUER;

/*****************************************************************************
 * VERIFIER QUE L'OI SPECIFIE EXISTE                                         *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   strcpy(identifiant_oi.classe, information_op_prim->information.communiquer.
          classe);
   strcpy(identifiant_oi.type, information_op_prim->information.communiquer.
          type);
   strcpy(identifiant_oi.nom, information_op_prim->information.communiquer.
          nom);
   identifiant_oi.num_copie = information_op_prim->information.communiquer.
                              num_copie;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OI_COMMUNIQUER;
   }

/*****************************************************************************
 * VERIFIER QUE LA BOITE OUT EXISTE                                          *
 *****************************************************************************/

if (*code_erreur == OK)
   {
```

```
   lecture_objet ("BOITE", "OUT", OBJET_INTERFACE, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = BOITE_OUT_EXISTE_PAS;
   }
}

/*****************************************************************************
 * TRAITEMENT D'UNE COMMUNICATION ORALE                                      *
 *****************************************************************************/

c_com_oral (information_op_prim, code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
ty_code_retour    cr;
ty_code_erreur    ce;
st_valeur_oi      info_oi;

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER QUE LE TELEPHONE EXISTE                                          *
 *****************************************************************************/

lecture_objet ("TELEPHONE", "TELEPHONE", OBJET_INTERFACE, &cr, &ce);

if (cr == EXISTE_PAS)
   *code_erreur = TELEPHONE_EXISTE_PAS;

/*****************************************************************************
 * VERIFIER QUE L'OI EXISTE                                                  *
 *****************************************************************************/

if (*code_erreur == OK)
   if (strcmp (information_op_prim->information.communiquer.classe, "") != 0)
      {
      strcpy (identifiant_oi.classe, information_op_prim->information.
              communiquer.classe);
      strcpy (identifiant_oi.type, information_op_prim->information.
              communiquer.type);
      strcpy (identifiant_oi.nom, information_op_prim->information.
              communiquer.nom);
      identifiant_oi.num_copie = information_op_prim->information.communiquer.
                                 num_copie;

      lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

      if (cr == EXISTE_PAS)
         *code_erreur = OI_COMMUNIQUER;
      }
}

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

```
controles_a_priori_consulter (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_consulter      *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_CONSULTER.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.consulter);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie =   ptr_courant->num_copie;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OBJET INFORMATIONNEL A CONSULTER                   *
 *****************************************************************************/

if ((strcmp(identifiant_oi.classe,"MESSAGE") != 0) AND
    (strcmp(identifiant_oi.classe,"FORMULAIRE") != 0) AND
    (strcmp(identifiant_oi.classe,"DOCUMENT") != 0))

{ *code_erreur = CLASSE_CONSULTER;
}

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL EXISTE                                *
 *****************************************************************************/

if (*code_erreur == OK)
{ lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

  if (code_retour_tmp == EXISTE_PAS)
  {  *code_erreur = OIE_CONSULTER;
  }
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_CONSULTER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_feuilleter (information_op_prim,code_erreur)
```

```
st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
char              classe [MX_LG_CLASSE + 1];
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_FEUILLETER.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OBJET INFORMATIONNEL                               *
 *****************************************************************************/

strcpy (classe, information_op_prim->information.feuilleter.classe);

if (strcmp (classe, "DOSSIER") != 0 AND
    strcmp (classe, "FICHIER") != 0 AND
    strcmp (classe, "PILE") != 0)

    *code_erreur = CLASSE_FEUILLETER;

/*****************************************************************************
 * VERIFIER QUE L'OIS EXISTE                                                 *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   strcpy (identifiant_oi.classe, information_op_prim->information.feuilleter.
           classe);
   strcpy (identifiant_oi.type, information_op_prim->information.feuilleter.
           type);
   strcpy (identifiant_oi.nom, information_op_prim->information.feuilleter.
           nom);
   identifiant_oi.num_copie = 0;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OIS_FEUILLETER;
   }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_FEUILLETER.\n");
#endif
}
```

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

controles_a_priori_enlever (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_ENLEVER.\n");
#endif

/*****************************************************************************
 * LA CLASSE DE L'OI A ETE VERIFIEE DANS LE MODULE OPERATION PRIMITIVE DE  *
 * MEME QUE SON EXISTENCE                                                  *
 *****************************************************************************/

*code_erreur = OK;

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_ENLEVER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_modifier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_MODIFIER.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * LA CLASSE DE L'OI A ETE VERIFIEE DANS LE MODULE OPERATION PRIMITIVE     *
 *****************************************************************************/

/*****************************************************************************
 * VERIFIER QUE LE DOCUMENT EXISTE                                         *
 *****************************************************************************/

strcpy (identifiant_oi.classe, "DOCUMENT");
strcpy (identifiant_oi.type, information_op_prim->information.modifier.type);
strcpy (identifiant_oi.nom, information_op_prim->information.modifier.nom);
```

```
identifiant_oi.num_copie =
information_op_prim->information.modifier.num_copie;

lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

if (cr == EXISTE_PAS)
   *code_erreur = OI_MODIFIER;

/*****************************************************************************
 * SI UN MESSAGE EST PRESENT , VERIFIER QU'IL EXISTE                         *
 *****************************************************************************/

if (*code_erreur == OK)
   if (strcmp (information_op_prim->information.modifier.nom_message,"") != 0)
      {
      strcpy (identifiant_oi.classe, "MESSAGE");
      strcpy (identifiant_oi.type  , "");
      strcpy (identifiant_oi.nom, information_op_prim->information.modifier.
              nom_message);
      identifiant_oi.num_copie = information_op_prim->information.modifier.
                                 num_copie_message;

      lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

      if (cr == EXISTE_PAS)
         *code_erreur = OI_MODIFIER;
      }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_MODIFIER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_completer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_completer      *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_COMPLETER.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.completer);
strcpy(identifiant_oi.classe,"FORMULAIRE");
```

```
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie = ptr_courant->num_copie;

/*****************************************************************************
 * LA CLASSE DE L'OI A ETE VERIFIEE DANS LE MODULE OPERATION PRIMITIVE      *
 *****************************************************************************/

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL EXISTE                                *
 *****************************************************************************/

lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

if (code_retour_tmp == EXISTE_PAS)
{  *code_erreur = OIE_COMPLETER;
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_COMPLETER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_verifier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
char              classe [MX_LG_CLASSE + 1];
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_VERIFIER.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OBJET INFORMATIONNEL A VERIFIER                   *
 *****************************************************************************/

strcpy (classe, information_op_prim->information.verifier.classe);

if (strcmp (classe, "FORMULAIRE") != 0 AND
    strcmp (classe, "DOSSIER") != 0)

   *code_erreur = CLASSE_VERIFIER;
```

```
/*****************************************************************************
 * VERIFIER QUE LE FORMULAIRE OU LE DOSSIER EXISTE                           *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   strcpy(identifiant_oi.classe, information_op_prim->information.verifier.
          classe);
   strcpy(identifiant_oi.type,information_op_prim->information.verifier.type);
   strcpy(identifiant_oi.nom, information_op_prim->information.verifier.nom);
   identifiant_oi.num_copie = information_op_prim->information.verifier.
                              num_copie;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OI_VERIFIER;
   }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_VERIFIER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_detruire (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_identifiant_oi id_pile_in;
   st_identifiant_oi id_pile_out;
   st_detruire       *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_DETRUIRE.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.detruire);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie = ptr_courant->num_copie;

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL A DETRUIRE EXISTE                     *
 *****************************************************************************/
```

```
lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

if (code_retour_tmp == EXISTE_PAS)
{  *code_erreur = OI_DETRUIRE;
}

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL N'EST PAS LA PILE DE LA BOITE IN    *
 *****************************************************************************/

if (*code_erreur == OK)
{   pile_objet_interface ("BOITE","IN",&id_pile_in,&code_retour_tmp,
                                                   &code_erreur_tmp);
    if ((strcmp(identifiant_oi.classe,id_pile_in.classe) == 0) AND
        (strcmp(identifiant_oi.type  ,id_pile_in.type) == 0) AND
        (strcmp(identifiant_oi.nom   ,id_pile_in.nom) == 0))

    {  *code_erreur = CLASSE_DETRUIRE;
    }
}

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL N'EST PAS LA PILE DE LA BOITE OUT   *
 *****************************************************************************/

if (*code_erreur == OK)
{  pile_objet_interface ("BOITE","OUT",&id_pile_out,&code_retour_tmp,
                                                    &code_erreur_tmp);
   if ((strcmp(identifiant_oi.classe,id_pile_out.classe) == 0) AND
       (strcmp(identifiant_oi.type  ,id_pile_out.type) == 0) AND
       (strcmp(identifiant_oi.nom   ,id_pile_out.nom) == 0))

   {  *code_erreur = CLASSE_DETRUIRE;
   }
}

/*****************************************************************************
 * VERIFIER QUE LA POUBELLE EXISTE                                          *
 *****************************************************************************/

if (*code_erreur == OK)
{  lecture_objet("POUBELLE","POUBELLE",OBJET_INTERFACE,&code_retour_tmp,
                                                       &code_erreur_tmp);
   if (code_retour_tmp == EXISTE_PAS)
   {  *code_erreur = POUBELLE_EXISTE_PAS;
   }
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_DETRUIRE.\n");
#endif
}
```

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

controles_a_priori_archiver (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
char              classe [MX_LG_CLASSE + 1];
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_ARCHIVER.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OI A ARCHIVER                                     *
 *****************************************************************************/

strcpy (classe, information_op_prim->information.archiver.classe);

if (strcmp (classe, "PILE") == 0 OR strcmp (classe, "MESSAGE") == 0)

   *code_erreur = CLASSE_ARCHIVER;

/*****************************************************************************
 * VERIFIER QUE L'OI A ARCHIVER EXISTE                                       *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   strcpy (identifiant_oi.classe, information_op_prim->information.archiver.
           classe);
   strcpy
(identifiant_oi.type,information_op_prim->information.archiver.type);
   strcpy (identifiant_oi.nom, information_op_prim->information.archiver.nom);
   identifiant_oi.num_copie = information_op_prim->information.archiver.
                              num_copie;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OI_ARCHIVER;
   }

/*****************************************************************************
 * VERIFIER QUE LA BOITE A ARCHIVE EXISTE                                    *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   lecture_objet ("BOITE", "ARCHIVE", OBJET_RANGEMENT, &cr, &ce);
```

```
   if (cr == EXISTE_PAS)
      *code_erreur = BOITE_ARCHIVE_EXISTE_PAS;
   }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_ARCHIVER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_trier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_trier          *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_TRIER.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.trier);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie = ptr_courant->num_copie;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OBJET INFORMATIONNEL A TRIER                      *
 *****************************************************************************/

if ((strcmp(identifiant_oi.classe,"DOSSIER") != 0) AND
    (strcmp(identifiant_oi.classe,"FICHIER") != 0) AND
    (strcmp(identifiant_oi.classe,"PILE") != 0))

{ *code_erreur = CLASSE_TRIER;
}

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL A TRIER EXISTE                        *
 *****************************************************************************/

if (*code_erreur == OK)
{  lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

   if (code_retour_tmp == EXISTE_PAS)
```

```
    {  *code_erreur = OIS_TRIER;
    }
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_TRIER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_classer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi       identifiant_oi;
st_valeur_oi            info_oi;
ty_code_retour          cr;
ty_code_erreur          ce;
char                    classe [MX_LG_CLASSE + 1];
char                    type   [MX_LG_TYPE + 1];
short                   trouve;
st_liste_identifiant_oi *liste_oi [1];
st_liste_identifiant_oi *ptr;
char                    type_oi [MX_LG_DESIR + 1];

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_CLASSER.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OI A CLASSE ET DE L'OBJET DE CLASSEMENT           *
 *****************************************************************************/

if (strcmp (information_op_prim->information.classer.classe, "PILE") == 0)

   *code_erreur = CLASSE_CLASSER;

/*****************************************************************************
 * VERIFIER QUE L'OIS EXISTE                                                 *
 *****************************************************************************/

if (*code_erreur == OK)
   {
   strcpy (identifiant_oi.classe, information_op_prim->information.classer.
           dest_classe);
   strcpy (identifiant_oi.type, information_op_prim->information.classer.
           dest_type);
   strcpy (identifiant_oi.nom, information_op_prim->information.classer.
           dest_nom);
```

```
   identifiant_oi.num_copie = 0;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OI_CLASSER;
   }

/*************************************************************************
 * VERIFIER QUE L'OBJET A CLASSER EXISTE                                 *
 *************************************************************************/

if (*code_erreur == OK)
   {
   strcpy(identifiant_oi.classe, information_op_prim->information.classer.
          classe);
   strcpy(identifiant_oi.type, information_op_prim->information.classer.type);
   strcpy(identifiant_oi.nom, information_op_prim->information.classer.nom);
   identifiant_oi.num_copie = information_op_prim->information.classer.
                              num_copie;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OI_CLASSER;
   }

/*************************************************************************
 * POUR UN FICHIER , L'OBJET QUE L'ON Y CLASSE DOIT ETRE DE MEME TYPE    *
 *************************************************************************/

if (*code_erreur == OK)
   if (strcmp (information_op_prim->information.classer.dest_classe,
     "FICHIER") == 0)
      {
      strcpy (identifiant_oi.classe, information_op_prim->information.classer.
              dest_classe);
      strcpy (identifiant_oi.type, information_op_prim->information.classer.
              dest_type);
      strcpy (identifiant_oi.nom, information_op_prim->information.classer.
              dest_nom);
      identifiant_oi.num_copie = 0;

      lire_type_fichier (&identifiant_oi, classe, type, &cr, &ce);

      if (cr != FICHIER_VIDE)
         if (strcmp (information_op_prim->information.classer.classe, classe)
             != 0
          OR strcmp (information_op_prim->information.classer.type,type)
             != 0)
             *code_erreur = ERREUR_FICHIER;
      }


/*************************************************************************
 *VERIFIER QUE L'OIS NE FAIT PAS PARTIE DES CONSTITUANTS DE L'OBJET A CLASSER*
 *************************************************************************/
```

```
if (*code_erreur == OK)
   {
   strcpy (identifiant_oi.classe, information_op_prim->information.classer.
                                  classe);
   strcpy (identifiant_oi.type, information_op_prim->information.classer.
                                type);
   strcpy (identifiant_oi.nom, information_op_prim->information.classer.
                               nom);
   identifiant_oi.num_copie = information_op_prim->information.classer.
                              num_copie;
   strcpy (type_oi, "OIS");

   constituant_ois_tout (&identifiant_oi, type_oi, liste_oi, &cr, &ce);

   trouve = FALSE;

   while (liste_oi [0] != NULL)
         {
         if (strcmp (liste_oi [0]->classe, information_op_prim->information.
                                           classer.dest_classe) == 0 OR
             strcmp (liste_oi [0]->type  , information_op_prim->information.
                                           classer.dest_type) == 0 OR
             strcmp (liste_oi [0]->nom   , information_op_prim->information.
                                           classer.dest_nom) == 0)
             trouve = TRUE;

         ptr = liste_oi [0];
         liste_oi [0] = liste_oi [0]->ptr_suivant;
         free (ptr);
         }
   if (trouve == TRUE)
      *code_erreur = OI_CLASSER;
   }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_CLASSER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_ranger (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_ranger         *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_CONTROLES_A_PRIORI
```

```
  printf(">>>CONTROLES_A_PRIORI_RANGER.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.ranger);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie = ptr_courant->num_copie;

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL A RANGER EXISTE                       *
 *****************************************************************************/

lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

if (code_retour_tmp == EXISTE_PAS)
{  *code_erreur = OI_RANGER;
}

/*****************************************************************************
 * VERIFIER QUE L'OBJET DE RANGEMENT EXISTE                                  *
 *****************************************************************************/

if (*code_erreur == OK)
{  lecture_objet(ptr_courant->dest_classe,ptr_courant->dest_nom,
                 OBJET_RANGEMENT,&code_retour_tmp,&code_erreur_tmp);

   if (code_retour_tmp == EXISTE_PAS)
   {  *code_erreur = RANGEMENT_EXISTE_PAS;
   }
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_RANGER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_replacer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
st_valeur_oi      info_oi;
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_REPLACER.\n");
```

```
#endif

*code_erreur = OK;

/*****************************************************************************
 * SI UN OI PRECIS EST SPECIFIE, ALORS IL FAUT QU'IL EXISTE                  *
 *****************************************************************************/

if (strcmp (information_op_prim->information.replacer.classe, "") != 0)
   {
   strcpy(identifiant_oi.classe, information_op_prim->information.replacer.
          classe);
   strcpy(identifiant_oi.type,information_op_prim->information.replacer.type);
   strcpy(identifiant_oi.nom, information_op_prim->information.replacer.nom);
   identifiant_oi.num_copie = information_op_prim->information.replacer.
                              num_copie;

   lecture_oi (&identifiant_oi, &info_oi, &cr, &ce);

   if (cr == EXISTE_PAS)
      *code_erreur = OI_REPLACER;
   }

/*****************************************************************************
 * SI UN OI EST SPECIFIE,ALORS IL FAUT TESTER QU'IL SE TROUVE SUR LE BUREAU*
 *****************************************************************************/

if (*code_erreur == OK)
   if (strcmp (information_op_prim->information.replacer.classe, "") != 0)
      {
      travail_oi (&identifiant_oi, &cr, &ce);

      if (cr != TABLE_DE_TRAVAIL AND cr != PHOTOCOPIEUSE)
         *code_erreur = PAS_SUR_OUTIL_TRAVAIL;
      }

#if DEBUG_CONTROLES_A_PRIORI
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_REPLACER.\n");
#endif
}

/*****************************************************************************/

controles_a_priori_decider (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   st_decider        *ptr_courant;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;
```

```
#if DEBUG_CONTROLES_A_PRIORI
  printf(">>>CONTROLES_A_PRIORI_DECIDER.\n");
#endif

*code_erreur = OK;

ptr_courant = &(information_op_prim->information.decider);
strcpy(identifiant_oi.classe,ptr_courant->classe);
strcpy(identifiant_oi.type  ,ptr_courant->type);
strcpy(identifiant_oi.nom   ,ptr_courant->nom);
identifiant_oi.num_copie = ptr_courant->num_copie;

/*****************************************************************************
 * VERIFIER LA CLASSE DE L'OBJET INFORMATIONNEL                               *
 *****************************************************************************/

if ((strcmp(identifiant_oi.classe,"FORMULAIRE") != 0) AND
    (strcmp(identifiant_oi.classe,"DOCUMENT") != 0) AND
    (strcmp(identifiant_oi.classe,"DOSSIER") != 0))

{ *code_erreur = CLASSE_DECIDER;
}

/*****************************************************************************
 * VERIFIER QUE L'OBJET INFORMATIONNEL EXISTE                                 *
 *****************************************************************************/

if (*code_erreur == OK)
{  lecture_oi (&identifiant_oi,&valeur_oi,&code_retour_tmp,&code_erreur_tmp);
   if (code_retour_tmp == EXISTE_PAS)
   {  *code_erreur = OI_DECIDER;
   }
}

#if DEBUG_CONTROLES_A_PRIORI
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONTROLES_A_PRIORI_DECIDER.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : graphique.h                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient les declarations du module de gestion */
/*               du graphisme.                                               */
/*                                                                           */
/*****************************************************************************/

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : graphique.c                                                     */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion du graphisme.      */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"
#include "graphique.h"

/*****************************************************************************/

graphique (information_op_prim,code_retour,code_erreur)

st_operation   *information_op_prim;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  char           type_operation [MX_TY_OPERATION + 1];
   ty_code_erreur code_erreur_tmp;

#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
   *code_retour = OK;

   strcpy (type_operation,information_op_prim->type_operation);

   if (strcmp (type_operation,"CREER") == 0)
   {  graphique_creer (information_op_prim,&code_erreur_tmp);
   }
   else if (strcmp (type_operation,"REPRODUIRE") == 0)
   {  graphique_reproduire (information_op_prim,&code_erreur_tmp);
   }
   else if (strcmp (type_operation,"RECEVOIR") == 0)
   {  graphique_recevoir (information_op_prim,&code_erreur_tmp);
   }
   else if (strcmp (type_operation,"COMMUNIQUER") == 0)
   {  graphique_communiquer (information_op_prim,&code_erreur_tmp);
   }
   else if (strcmp (type_operation,"CONSULTER") == 0)
   {  graphique_consulter (information_op_prim,&code_erreur_tmp);
   }
   else if (strcmp (type_operation,"FEUILLETER") == 0)
   {  graphique_feuilleter (information_op_prim,&code_erreur_tmp);
   }
   else if (strcmp (type_operation,"ENLEVER") == 0)
   {  graphique_enlever (information_op_prim,&code_erreur_tmp);
```

```
    }
    else if (strcmp (type_operation,"MODIFIER") == 0)
    {  graphique_modifier (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"COMPLETER") == 0)
    {  graphique_completer (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"VERIFIER") == 0)
    {  graphique_verifier (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"DETRUIRE") == 0)
    {  graphique_detruire (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"ARCHIVER") == 0)
    {  graphique_archiver (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"TRIER") == 0)
    {  graphique_trier (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"CLASSER") == 0)
    {  graphique_classer (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"RANGER") == 0)
    {  graphique_ranger (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"REPLACER") == 0)
    {  graphique_replacer (information_op_prim,&code_erreur_tmp);
    }
    else if (strcmp (type_operation,"DECIDER") == 0)
    {  graphique_decider (information_op_prim,&code_erreur_tmp);
    }

    if (code_erreur_tmp != OK)
    {  message_erreur (code_erreur_tmp);
       *code_erreur = ERREUR;
    }

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE.\n");
#endif
}

/*****************************************************************************/

graphique_creer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_CREER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
```

```
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_CREER.\n");
#endif
}

/*****************************************************************************/

graphique_reproduire (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_REPRODUIRE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_REPRODUIRE.\n");
#endif
}

/*****************************************************************************/

graphique_recevoir (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_RECEVOIR.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
```

```
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_RECEVOIR.\n");
#endif
}

/*****************************************************************************/

graphique_communiquer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_COMMUNIQUER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_COMMUNIQUER.\n");
#endif
}

/*****************************************************************************/

graphique_consulter (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_CONSULTER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_CONSULTER.\n");
#endif
}

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

```
graphique_feuilleter (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_FEUILLETER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_FEUILLETER.\n");
#endif
}

/*************************************************************************/

graphique_enlever (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_ENLEVER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_ENLEVER.\n");
#endif
}

/*************************************************************************/

graphique_modifier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
```

```
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_MODIFIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_MODIFIER.\n");
#endif
}

/*****************************************************************************/

graphique_completer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_COMPLETER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_COMPLETER.\n");
#endif
}

/*****************************************************************************/

graphique_verifier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{ ty_code_retour cr;
  ty_code_erreur ce;

#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_VERIFIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
```

```
#endif

   *code_erreur = OK;

assignation_parametre ("parametre1","COMPLET",&cr,&ce);
assignation_parametre ("parametre2","COMPLET",&cr,&ce);

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_VERIFIER.\n");
#endif
}

/*****************************************************************************/

graphique_detruire (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_DETRUIRE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_DETRUIRE.\n");
#endif
}

/*****************************************************************************/

graphique_archiver (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_ARCHIVER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;
```

```
#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_ARCHIVER.\n");
#endif
}

/*****************************************************************************/

graphique_trier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_TRIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_TRIER.\n");
#endif
}

/*****************************************************************************/

graphique_classer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_CLASSER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_CLASSER.\n");
#endif
}
```

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

graphique_ranger (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_RANGER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_RANGER.\n");
#endif
}

/*****************************************************************************/

graphique_replacer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_REPLACER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

   *code_erreur = OK;

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_REPLACER.\n");
#endif
}

/*****************************************************************************/

graphique_decider (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;
```

```
{ ty_code_retour cr;
  ty_code_erreur ce;

#if DEBUG_GRAPHIQUE
  printf(">>>GRAPHIQUE_DECIDER.\n");
  printf("-------------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------------\n");
#endif

   *code_erreur = OK;

assignation_parametre ("parametre3","EXAMEN ENTREE",&cr,&ce);

#if DEBUG_GRAPHIQUE
  printf("-------------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------------\n");
  printf("<<<GRAPHIQUE_DECIDER.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : manip_bd.c                                                      */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion de manipulation    */
/*               de la base de donnees.                                      */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

manipulation_bd (information_op_prim,code_retour,code_erreur)

st_operation   *information_op_prim;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  char           type_operation [MX_TY_OPERATION + 1];
   ty_code_erreur code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD.\n");
  printf("-----------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-----------------------------------------------------\n");
#endif

*code_erreur = OK;
*code_retour = OK;

strcpy (type_operation,information_op_prim->type_operation);

if (strcmp (type_operation,"CREER") == 0)
{  manipulation_bd_creer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"REPRODUIRE") == 0)
{  manipulation_bd_reproduire (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"RECEVOIR") == 0)
{  manipulation_bd_recevoir (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"COMMUNIQUER") == 0)
{  manipulation_bd_communiquer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"CONSULTER") == 0)
{  manipulation_bd_consulter (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"FEUILLETER") == 0)
{  manipulation_bd_feuilleter (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"ENLEVER") == 0)
{  manipulation_bd_enlever (information_op_prim,&code_erreur_tmp);
```

```
}
else if (strcmp (type_operation,"MODIFIER") == 0)
{  manipulation_bd_modifier (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"COMPLETER") == 0)
{  manipulation_bd_completer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"VERIFIER") == 0)
{  manipulation_bd_verifier (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"DETRUIRE") == 0)
{  manipulation_bd_detruire (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"ARCHIVER") == 0)
{  manipulation_bd_archiver (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"TRIER") == 0)
{  manipulation_bd_trier (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"CLASSER") == 0)
{  manipulation_bd_classer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"RANGER") == 0)
{  manipulation_bd_ranger (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"REPLACER") == 0)
{  manipulation_bd_replacer (information_op_prim,&code_erreur_tmp);
}
else if (strcmp (type_operation,"DECIDER") == 0)
{  manipulation_bd_decider (information_op_prim,&code_erreur_tmp);
}

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code retour %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_creer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_CREER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
```

```
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*************************************************************************
 * ON CREE L'OI SPECIFIE                                                 *
 *************************************************************************/

strcpy(valeur_oi.classe,information_op_prim->information.creer.classe);
strcpy(valeur_oi.type,information_op_prim->information.creer.type);
strcpy(valeur_oi.nom,information_op_prim->information.creer.nom);
valeur_oi.num_copie = 0;
strcpy(valeur_oi.etat,information_op_prim->information.creer.etat);
strcpy(valeur_oi.echeance,information_op_prim->information.creer.echeance);
strcpy(valeur_oi.ordre_clas,information_op_prim->information.creer.ordre_clas)
;
strcpy(valeur_oi.etat_contenu,"VIDE");

creer_oi(&valeur_oi,&code_retour_tmp,&code_erreur_tmp);

/*************************************************************************
 * ON ASSOCIE CET OI A LA TABLE DE TRAVAIL                               *
 *************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.creer.classe);
strcpy(identifiant_oi.type,information_op_prim->information.creer.type);
strcpy(identifiant_oi.nom,information_op_prim->information.creer.nom);
identifiant_oi.num_copie = 0;

outil_travail_oi(&identifiant_oi,"TABLE_DE_TRAVAIL",&code_retour_tmp,
                                                    &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_CREER.\n");
#endif
}

/*************************************************************************/

manipulation_bd_reproduire (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
int               numero_copie;
st_valeur_oi      valeur_oi;
st_identifiant_oi identifiant_oi;
ty_code_retour    cr;
ty_code_erreur    ce;
short             existe;
short             lieu_original;
```

```
#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_REPRODUIRE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*************************************************************************
 * 1. TRAITEMENT DES COPIES                                              *
 *************************************************************************/

numero_copie = information_op_prim->information.reproduire.num_depart;

while (numero_copie < information_op_prim->information.reproduire.num_depart +
                      information_op_prim->information.reproduire.nbr_copie)
      {

      /*************************************************************************
       * 1.a CREATION DE LA COPIE                                              *
       *************************************************************************/

      strcpy (valeur_oi.classe, information_op_prim->information.reproduire.
                                classe);
      strcpy (valeur_oi.type, information_op_prim->information.reproduire.
                              type);
      strcpy (valeur_oi.nom, information_op_prim->information.reproduire.
                             nom);
      valeur_oi.num_copie = numero_copie;

      valeur_oi.etat [0] = '\0';
      valeur_oi.echeance [0] = '\0';
      valeur_oi.ordre_clas [0] = '\0';
      valeur_oi.etat_contenu [0] ='\0';

      creer_oi (&valeur_oi, &cr, &ce);

      /*************************************************************************
       * 1.b ASSOCIER LA COPIE CREEE A LA PHOTOCOPIEUSE                        *
       *************************************************************************/

      strcpy (identifiant_oi.classe, information_op_prim->information.
                                      reproduire.classe);
      strcpy (identifiant_oi.type, information_op_prim->information.
                                   reproduire.type);
      strcpy (identifiant_oi.nom, information_op_prim->information.
                                  reproduire.nom);
      identifiant_oi.num_copie = numero_copie;

      outil_travail_oi (&identifiant_oi, "PHOTOCOPIEUSE", &cr, &ce);

      numero_copie++;
      }
```

```
/*****************************************************************************
 * 2. TRAITEMENT DE L'ORIGINAL                                               *
 *****************************************************************************/

/*****************************************************************************
 * 2.a VERIFIER QUE L'ORIGINAL EXISTE                                        *
 *****************************************************************************/

identifiant_oi.num_copie = information_op_prim->information.reproduire.
                           num_copie;

lecture_oi (&identifiant_oi, &valeur_oi, &cr, &ce);

if (cr == EXISTE_PAS)
   existe = FALSE;
   else existe = TRUE;

/*****************************************************************************
 * 2.b REGARDER OU SE TROUVE L'ORIGINAL                                      *
 *****************************************************************************/

lieu_original = OK;
if (existe == TRUE)
   {
   travail_oi (&identifiant_oi, &cr, &ce);

   if (cr == PHOTOCOPIEUSE)
      lieu_original = PHOTOCOPIEUSE;
      else if (cr == TABLE_DE_TRAVAIL)
              lieu_original = TABLE_DE_TRAVAIL;
   }

/*****************************************************************************
 * 2.c SI L'ORIGINAL SE TROUVE SUR LA TABLE DE TRAVAIL,IL FAUT L'ENLEVER*
 *****************************************************************************/

if (lieu_original == TABLE_DE_TRAVAIL)

   replacer_oi (&identifiant_oi, &cr, &ce);

/*****************************************************************************
 * 2.d ASSOCIER L'ORIGINAL A LA PHOTOCOPIEUSE                                *
 *****************************************************************************/

if (existe == TRUE AND lieu_original != PHOTOCOPIEUSE)

   outil_travail_oi (&identifiant_oi, "PHOTOCOPIEUSE", &cr, &ce);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_REPRODUIRE.\n");
#endif
}
```

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

manipulation_bd_recevoir (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi  identifiant_oi;
   st_identifiant_oi  identifiant_ois;
   st_identifiant_oi  id_comp;
   st_valeur_oi       valeur_oi;
   st_valeur_oi       val_comp;
   st_liste_valeur_oi *ptr_composant;
   st_liste_valeur_oi *ptr_tmp;
   char               mode_reception [MX_LG_MODE + 1];
   ty_code_retour     code_retour_tmp;
   ty_code_erreur     code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_RECEVOIR.\n");
  printf("-----------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-----------------------------------------------------\n");
#endif

*code_erreur = OK;

strcpy(mode_reception,information_op_prim->information.recevoir.mode_reception
);

/*****************************************************************************
 * ON CREE L'OI SPECIFIE (SAUF RECEPTION MESSAGE ORAL SANS TRACE)       *
 *****************************************************************************/

if (strcmp(information_op_prim->information.recevoir.classe,"") != 0)
{
   strcpy(valeur_oi.classe,information_op_prim->information.recevoir.classe);
   strcpy(valeur_oi.type,information_op_prim->information.recevoir.type);
   strcpy(valeur_oi.nom,information_op_prim->information.recevoir.nom);
   valeur_oi.num_copie = information_op_prim->information.recevoir.num_copie;;
   strcpy(valeur_oi.etat,information_op_prim->information.recevoir.etat);
   strcpy(valeur_oi.echeance,
                        information_op_prim->information.recevoir.echeance);
   strcpy(valeur_oi.ordre_clas,
                        information_op_prim->information.recevoir.ordre_clas);
   strcpy(valeur_oi.etat_contenu,"VIDE");

   creer_oi(&valeur_oi,&code_retour_tmp,&code_erreur_tmp);
}

/*****************************************************************************
 * ON ASSSOCIE CET OI SOIT AU TELEPHONE SOIT A LA PILE DE LA BOITE IN   *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.recevoir.classe)
```

```
strcpy(identifiant_oi.type,information_op_prim->information.recevoir.type);
strcpy(identifiant_oi.nom,information_op_prim->information.recevoir.nom);
identifiant_oi.num_copie =information_op_prim->information.recevoir.num_copie;

if (strcmp(mode_reception,"ECRIT") == 0)
{  pile_objet_interface("BOITE","IN",&identifiant_ois,&code_retour_tmp,
                                                    &code_erreur_tmp);
   classer_oi(&identifiant_oi,&identifiant_ois,&code_retour_tmp,
                                                &code_erreur_tmp);
}
else
{  interface_oi(&identifiant_oi,"TELEPHONE","TELEPHONE",&code_retour_tmp,
                                                      &code_erreur_tmp);
}

/*************************************************************************
 * ON CREE CHACUN DES COMPOSANTS ET ON LES CLASSE DANS L'OI              *
 *************************************************************************/

ptr_composant = information_op_prim->information.recevoir.composants;

while (ptr_composant != NULL)
{
   strcpy(val_comp.classe,ptr_composant->classe);
   strcpy(val_comp.type,ptr_composant->type);
   strcpy(val_comp.nom,ptr_composant->nom);
   val_comp.num_copie = ptr_composant->num_copie;;
   strcpy(val_comp.etat,ptr_composant->etat);
   strcpy(val_comp.echeance,ptr_composant->echeance);
   strcpy(val_comp.ordre_clas,"");
   strcpy(val_comp.etat_contenu,"VIDE");

   creer_oi(&val_comp,&code_retour_tmp,&code_erreur_tmp);

   strcpy(id_comp.classe,ptr_composant->classe);
   strcpy(id_comp.type,ptr_composant->type);
   strcpy(id_comp.nom,ptr_composant->nom);
   id_comp.num_copie = ptr_composant->num_copie;;

   classer_oi(&id_comp,&identifiant_oi,&code_retour_tmp,&code_erreur_tmp);

   ptr_tmp = ptr_composant;
   ptr_composant = ptr_composant->ptr_suivant;
   free (ptr_tmp);
}

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_RECEVOIR.\n");
#endif
}

/*************************************************************************/

manipulation_bd_communiquer (information_op_prim,code_erreur)
```

```
st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
ty_code_retour    cr;
ty_code_erreur    ce;
st_identifiant_oi identifiant_oi;
st_identifiant_oi id_pile;
int               b_etat, b_echeance, b_contenu, b_ordre;
st_valeur_oi      valeur_oi;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_COMMUNIQUER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*************************************************************************
 * REMARQUE : UNE COMMUNICATION ORALE N'ENTRAINE PAS DE MODIFICATIONS   *
 *************************************************************************/

/*************************************************************************
 * TRAITEMENT D'UNE COMMUNICATION ECRITE                                 *
 *************************************************************************/

if (strcmp (information_op_prim->information.communiquer.mode_communication,
            "ORAL") != 0)
   {
   /*************************************************************************
    * a. DETACHER L'OI A COMMUNIQUER DE TOUT OBJET DU BUREAU                *
    *************************************************************************/

   strcpy(identifiant_oi.classe, information_op_prim->information.communiquer.
                                 classe);
   strcpy(identifiant_oi.type, information_op_prim->information.communiquer.
                               type);
   strcpy(identifiant_oi.nom, information_op_prim->information.communiquer.
                              nom);
   identifiant_oi.num_copie = information_op_prim->information.communiquer.
                              num_copie;

   detacher_oi (&identifiant_oi, &cr, &ce);

   /*************************************************************************
    * b. TROUVER L'IDENTIFIANT DE LA PILE ASSOCIEE A LA BOITE OUT           *
    *************************************************************************/

   pile_objet_interface ("BOITE", "OUT", &id_pile, &cr, &ce);
```

```
    /*************************************************************************
     * c. ASSOCIER L'OI A LA PILE                                            *
     *************************************************************************/

    classer_oi (&identifiant_oi, &id_pile, &cr, &ce);

    /*************************************************************************
     * d. MODIFICATION EVENTUELLE DE L'ETAT ET DE L'ECHEANCE                 *
     *************************************************************************/

    b_etat = b_echeance = b_ordre = b_contenu = FAUX;

    if (strcmp(information_op_prim->information.communiquer.etat, "") != 0)
       b_etat = VRAI;

    if (strcmp(information_op_prim->information.communiquer.echeance, "") != 0)
       b_echeance = VRAI;

    if (b_etat == VRAI OR b_echeance == VRAI)
       {
       strcpy(valeur_oi.classe, information_op_prim->information.communiquer.
                                classe);
       strcpy(valeur_oi.type, information_op_prim->information.communiquer.
                              type);
       strcpy(valeur_oi.nom, information_op_prim->information.communiquer.
                             nom);
       strcpy(valeur_oi.etat, information_op_prim->information.communiquer.
                              etat);
       strcpy(valeur_oi.echeance, information_op_prim->information.communiquer.
                                  echeance);
       valeur_oi.num_copie = information_op_prim->information.communiquer.
                             num_copie;

       modifier_oi (&valeur_oi, &b_etat, &b_echeance, &b_contenu, &b_ordre,
                    &cr, &ce);
       }
    }

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_COMMUNIQUER.\n");
#endif
}

/*************************************************************************/

manipulation_bd_consulter (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   int               b_etat;
   int               b_echeance;
```

```
   int              b_contenu;
   int              b_ordre;
   ty_code_retour   code_retour_tmp;
   ty_code_erreur   code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_CONSULTER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*************************************************************************
 * ON MET A JOUR LES ATTRIBUTS ETAT ET ECHEANCE DE L'OI                  *
 *************************************************************************/

strcpy(valeur_oi.classe,information_op_prim->information.consulter.classe);
strcpy(valeur_oi.type,  information_op_prim->information.consulter.type);
strcpy(valeur_oi.nom,   information_op_prim->information.consulter.nom);
valeur_oi.num_copie =   information_op_prim->information.consulter.num_copie;
strcpy(valeur_oi.etat,  information_op_prim->information.consulter.etat);
strcpy(valeur_oi.echeance,
                        information_op_prim->information.consulter.echeance);
strcpy(valeur_oi.ordre_clas,"");
strcpy(valeur_oi.etat_contenu,"VIDE");

b_etat     = VRAI;
b_echeance = VRAI;
b_contenu  = FAUX;
b_ordre    = FAUX;

modifier_oi(&valeur_oi,&b_etat,&b_echeance,&b_contenu,&b_ordre,
                            &code_retour_tmp,&code_erreur_tmp);

/*************************************************************************
 * ON ASSOCIE CET OI A LA TABLE DE TRAVAIL                               *
 *************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.consulter.classe
);
strcpy(identifiant_oi.type,information_op_prim->information.consulter.type);
strcpy(identifiant_oi.nom,information_op_prim->information.consulter.nom);
identifiant_oi.num_copie =
information_op_prim->information.consulter.num_copie;

outil_travail_oi(&identifiant_oi,"TABLE_DE_TRAVAIL",&code_retour_tmp,
                                                    &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_CONSULTER.\n");
```

```
#endif
}

/*****************************************************************************/

manipulation_bd_feuilleter (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_ois;
st_valeur_oi      valeur_oi;
ty_code_retour    cr;
ty_code_erreur    ce;
short             endroit_ois;
int               b_ordre, b_echeance, b_etat, b_contenu;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_FEUILLETER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * LES OI CLASSE DANS L'OIS SE TROUVANT SUR UN OUTIL DE TRAVAIL, NE DOIVENT *
 * PAS ETRE VISUALISES !!!!!!!                                              *
 *****************************************************************************/

/*****************************************************************************
 * a. TESTER SI L'OIS SE TROUVE SUR LE BUREAU                               *
 *****************************************************************************/

strcpy (identifiant_ois.classe, information_op_prim->information.feuilleter.
                                 classe);
strcpy (identifiant_ois.nom, information_op_prim->information.feuilleter.
                             nom);
strcpy (identifiant_ois.type, information_op_prim->information.feuilleter.
                              type);
identifiant_ois.num_copie = 0;

travail_oi (&identifiant_ois, &cr, &ce);

endroit_ois = cr;

/*****************************************************************************
 * b. PLACER L'OIS SUR LA TABLE DE TRAVAIL S'IL NE S'Y TROUVE PAS DEJA      *
 *****************************************************************************/

if (endroit_ois == PHOTOCOPIEUSE)

   replacer_oi (&identifiant_ois, &cr, &ce);
```

```
if (endroit_ois == PHOTOCOPIEUSE OR endroit_ois == OK)

   outil_travail_oi (&identifiant_ois, "TABLE_DE_TRAVAIL", &cr, &ce);


/*****************************************************************************
* c. MODIFICATION EVENTUELLE DE L'ETAT ET DE L'ECHEANCE                      *
*****************************************************************************/

b_etat = b_echeance = b_ordre = b_contenu = FAUX;

if (strcmp (information_op_prim->information.feuilleter.etat, "") != 0)
   b_etat = VRAI;

if (strcmp (information_op_prim->information.feuilleter.echeance, "") != 0)
   b_echeance = VRAI;

if (b_etat == VRAI OR b_echeance == VRAI)
   {
   strcpy (valeur_oi.classe, information_op_prim->information.feuilleter.
                             classe);
   strcpy (valeur_oi.type, information_op_prim->information.feuilleter.
                           type);
   strcpy (valeur_oi.nom, information_op_prim->information.feuilleter.
                          nom);
   strcpy (valeur_oi.etat, information_op_prim->information.feuilleter.
                           etat);
   strcpy (valeur_oi.echeance, information_op_prim->information.feuilleter.
                               echeance);
   valeur_oi.num_copie = 0;

   modifier_oi (&valeur_oi, &b_etat, &b_echeance, &b_contenu, &b_ordre,
                &cr, &ce);
   }

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_FEUILLETER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_enlever (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_ENLEVER.\n");
```

```
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * ON DETACHE L'OI                                                           *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.enlever.classe);
strcpy(identifiant_oi.type,information_op_prim->information.enlever.type);
strcpy(identifiant_oi.nom,information_op_prim->information.enlever.nom);
identifiant_oi.num_copie = information_op_prim->information.enlever.num_copie;

detacher_oi(&identifiant_oi,&code_retour_tmp,&code_erreur_tmp);

/*****************************************************************************
 * ON ASSOCIE CET OI A LA TABLE DE TRAVAIL                                   *
 *****************************************************************************/

outil_travail_oi(&identifiant_oi,"TABLE_DE_TRAVAIL",&code_retour_tmp,
                                                    &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_ENLEVER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_modifier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
short             endroit_oi;
ty_code_retour    cr;
ty_code_erreur    ce;
st_valeur_oi      valeur_oi;
int               b_etat, b_echeance, b_ordre, b_contenu;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_MODIFIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
```

```
#endif

*code_erreur = OK;

/*****************************************************************************
 * a. REGARDER OU SE TROUVE LE DOCUMENT                                      *
 *****************************************************************************/

strcpy (identifiant_oi.classe, "DOCUMENT");
strcpy (identifiant_oi.type, information_op_prim->information.modifier.type);
strcpy (identifiant_oi.nom, information_op_prim->information.modifier.nom);
identifiant_oi.num_copie =information_op_prim->information.modifier.num_copie;

travail_oi (&identifiant_oi, &cr, &ce);

endroit_oi = cr;

/*****************************************************************************
 * b. PLACER LE DOCUMENT SUR LE BUREAU S'IL NE S'Y TROUVE PAS                *
 *****************************************************************************/

if (endroit_oi == PHOTOCOPIEUSE)

   replacer_oi (&identifiant_oi, &cr, &ce);

if (endroit_oi == PHOTOCOPIEUSE OR endroit_oi == OK)

   outil_travail_oi (&identifiant_oi, "TABLE_DE_TRAVAIL", &cr, &ce);

/*****************************************************************************
 * c. MODIFIER LE DOCUMENT : SEUL L'ETAT ET L'ECHEANCE PEUVENT ETRE MODIFIES*
 *****************************************************************************/

b_ordre = b_contenu = b_etat = b_echeance = FAUX;

if (strcmp (information_op_prim->information.modifier.etat, "") != 0)
   b_etat = VRAI;

if (strcmp (information_op_prim->information.modifier.echeance, "") != 0)
   b_echeance = VRAI;

if (b_etat == VRAI OR b_echeance == VRAI)
   {
   strcpy (valeur_oi.classe, "DOCUMENT");
   strcpy (valeur_oi.type, information_op_prim->information.modifier.
                           type);
   strcpy (valeur_oi.nom, information_op_prim->information.modifier.
                          nom);
   strcpy (valeur_oi.etat, information_op_prim->information.modifier.
                           etat);
   strcpy (valeur_oi.echeance, information_op_prim->information.modifier.
                               echeance);
   valeur_oi.num_copie = information_op_prim->information.modifier.num_copie;

   modifier_oi (&valeur_oi, &b_etat, &b_echeance, &b_contenu, &b_ordre,
                &cr, &ce);
   }
```

```
/*****************************************************************************
 * d. SI UN MESSAGE EST SPECIFIE, REGARDER OU IL SE TROUVE                   *
 *****************************************************************************/

if (strcmp (information_op_prim->information.modifier.nom_message, "") != 0)
   {
   strcpy (identifiant_oi.classe, "MESSAGE");
   strcpy (identifiant_oi.nom, information_op_prim->information.modifier.
                               nom_message);
   strcpy (identifiant_oi.type, "");
   identifiant_oi.num_copie = information_op_prim->information.modifier.
                              num_copie_message;

   travail_oi (&identifiant_oi, &cr, &ce);

   endroit_oi = cr;

   /*************************************************************************
    * d.1 PLACER LE MESSAGE SUR LA TABLE DE TRAVAIL S'IL N'Y EST PAS DEJA   *
    *************************************************************************/

   if (endroit_oi == PHOTOCOPIEUSE)

      replacer_oi (&identifiant_oi, &cr, &ce);

   if (endroit_oi == PHOTOCOPIEUSE OR endroit_oi == OK)

      outil_travail_oi (&identifiant_oi, "TABLE_DE_TRAVAIL", &cr, &ce);
   }

#if DEBUG_MANIPULATION_BD
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_MODIFIER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_completer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   int               b_etat;
   int               b_echeance;
   int               b_contenu;
   int               b_ordre;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_COMPLETER.\n");
  printf("-----------------------------------------------------\n");
```

```
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-----------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * ON MET A JOUR LES ATTRIBUTS ETAT ET ECHEANCE DE L'OI                      *
 *****************************************************************************/

strcpy(valeur_oi.classe,"FORMULAIRE");
strcpy(valeur_oi.type,  information_op_prim->information.completer.type);
strcpy(valeur_oi.nom,   information_op_prim->information.completer.nom);
valeur_oi.num_copie =   information_op_prim->information.completer.num_copie;
strcpy(valeur_oi.etat,  information_op_prim->information.completer.etat);
strcpy(valeur_oi.echeance,
                        information_op_prim->information.completer.echeance);
strcpy(valeur_oi.ordre_clas,"");
strcpy(valeur_oi.etat_contenu,"VIDE");

b_etat     = VRAI;
b_echeance = VRAI;
b_contenu  = FAUX;
b_ordre    = FAUX;

modifier_oi(&valeur_oi,&b_etat,&b_echeance,&b_contenu,&b_ordre,
                                          &code_retour_tmp,&code_erreur_tmp);

/*****************************************************************************
 * ON ASSOCIE CET OI A LA TABLE DE TRAVAIL                                   *
 *****************************************************************************/

strcpy(identifiant_oi.classe,"FORMULAIRE");
strcpy(identifiant_oi.type,information_op_prim->information.completer.type);
strcpy(identifiant_oi.nom,information_op_prim->information.completer.nom);
identifiant_oi.num_copie =
                        information_op_prim->information.completer.num_copie;

outil_travail_oi(&identifiant_oi,"TABLE_DE_TRAVAIL",&code_retour_tmp,
                                                &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-----------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_COMPLETER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_verifier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;
```

```
{
st_identifiant_oi identifiant_oi;
ty_code_retour    cr;
ty_code_erreur    ce;
short             endroit_oi;
int               b_ordre, b_echeance, b_etat, b_contenu;
st_valeur_oi      valeur_oi;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_VERIFIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * a. REGARDER OU SE TROUVE L'OBJET INFORMATIONNEL QUE L'ON VA VERIFIER   *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.verifier.classe)
strcpy(identifiant_oi.type,information_op_prim->information.verifier.type);
strcpy(identifiant_oi.nom,information_op_prim->information.verifier.nom);
identifiant_oi.num_copie =information_op_prim->information.verifier.num_copie;

travail_oi (&identifiant_oi, &cr, &ce);

endroit_oi = cr;

/*****************************************************************************
 * b. PLACER L'OI SUR LA TABLE DE TRAVAIL                                    *
 *****************************************************************************/

if (endroit_oi == PHOTOCOPIEUSE)

   replacer_oi (&identifiant_oi, &cr, &ce);

if (endroit_oi == PHOTOCOPIEUSE OR endroit_oi == OK)

   outil_travail_oi (&identifiant_oi, "TABLE_DE_TRAVAIL", &cr, &ce);

/*****************************************************************************
 * c. MODIFIER L'ETAT ET L'ECHEANCE DE L'OI                                  *
 *****************************************************************************/

b_ordre = b_etat = b_echeance = b_contenu = FAUX;

if (strcmp (information_op_prim->information.verifier.etat, "") != 0)
   b_etat = VRAI;

if (strcmp (information_op_prim->information.verifier.echeance, "") != 0)
   b_echeance = VRAI;
```

```
if (b_etat == VRAI OR b_echeance == VRAI)
   {
   strcpy (valeur_oi.classe,
information_op_prim->information.verifier.classe);
   strcpy (valeur_oi.type, information_op_prim->information.verifier.type);
   strcpy (valeur_oi.nom, information_op_prim->information.verifier.nom);
   strcpy (valeur_oi.etat, information_op_prim->information.verifier.etat);
   strcpy (valeur_oi.echeance, information_op_prim->information.verifier.
                               echeance);
   strcpy (valeur_oi.ordre_clas, "");
   strcpy (valeur_oi.etat_contenu, "");
   valeur_oi.num_copie = information_op_prim->information.verifier.num_copie;

   modifier_oi (&valeur_oi, &b_etat, &b_echeance, &b_contenu, &b_ordre, &cr,
                &ce);
   }

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_VERIFIER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_detruire (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   int               b_etat;
   int               b_echeance;
   int               b_contenu;
   int               b_ordre;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_DETRUIRE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * ON MET A JOUR LES ATTRIBUTS ETAT ET ECHEANCE DE L'OI                      *
 *****************************************************************************/

strcpy(valeur_oi.classe,information_op_prim->information.detruire.classe);
```

```
strcpy(valeur_oi.type,  information_op_prim->information.detruire.type);
strcpy(valeur_oi.nom,   information_op_prim->information.detruire.nom);
valeur_oi.num_copie =   information_op_prim->information.detruire.num_copie;
strcpy(valeur_oi.etat,  information_op_prim->information.detruire.etat);
strcpy(valeur_oi.echeance,
                        information_op_prim->information.detruire.echeance);
strcpy(valeur_oi.ordre_clas,"");
strcpy(valeur_oi.etat_contenu,"VIDE");

b_etat     = VRAI;
b_echeance = VRAI;
b_contenu  = FAUX;
b_ordre    = FAUX;

modifier_oi(&valeur_oi,&b_etat,&b_echeance,&b_contenu,&b_ordre,
                               &code_retour_tmp,&code_erreur_tmp);

/*****************************************************************************
 * ON DETACHE L'OI                                                           *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.detruire.classe)
strcpy(identifiant_oi.type,information_op_prim->information.detruire.type);
strcpy(identifiant_oi.nom,information_op_prim->information.detruire.nom);
identifiant_oi.num_copie =information_op_prim->information.detruire.num_copie;

detacher_oi(&identifiant_oi,&code_retour_tmp,&code_erreur_tmp);

/*****************************************************************************
 * ON ASSOCIE L'OI A LA POUBELLE                                             *
 *****************************************************************************/

interface_oi(&identifiant_oi,"POUBELLE","POUBELLE",&code_retour_tmp,
                                                   &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_DETRUIRE.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_archiver (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
ty_code_retour    cr;
ty_code_erreur    ce;
int               b_ordre, b_etat, b_echeance, b_contenu;
st_valeur_oi      valeur_oi;
```

```
#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_ARCHIVER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * a. SI ON EST EN PRESENCE D'UN OIS, IL FAUT ARCHIVER TOUS SES CONSTITUANTS*
 *****************************************************************************/

strcpy (identifiant_oi.classe,information_op_prim->information.archiver.
                              classe);
strcpy (identifiant_oi.type  ,information_op_prim->information.archiver.type);
strcpy (identifiant_oi.nom   ,information_op_prim->information.archiver.nom);
identifiant_oi.num_copie =information_op_prim->information.archiver.num_copie;

if (strcmp   (information_op_prim->information.archiver.classe,"DOSSIER")== 0
   OR strcmp (information_op_prim->information.archiver.classe,"PILE")   == 0
   OR strcmp (information_op_prim->information.archiver.classe,"FICHIER")== 0)

   replacer_const (&identifiant_oi, &cr, &ce);

/*****************************************************************************
 * b. DETACHER L'OI DE TOUTE ASSOCIATION                                     *
 *****************************************************************************/

detacher_oi (&identifiant_oi, &cr, &ce);

/*****************************************************************************
 * c. ASSOCIER L'OI A LA BOITE A ARCHIVES                                    *
 *****************************************************************************/

ranger_oi (&identifiant_oi, "BOITE", "ARCHIVE", &cr, &ce);

/*****************************************************************************
 * d. MODIFIER L'ETAT ET L'ECHEANCE DE L'OI                                  *
 *****************************************************************************/

b_ordre = b_etat = b_echeance = b_contenu = FAUX;

if (strcmp (information_op_prim->information.archiver.etat, "") != 0)
   b_etat = VRAI;

if (strcmp (information_op_prim->information.archiver.echeance, "") != 0)
   b_echeance = VRAI;

if (b_etat == VRAI OR b_echeance == VRAI)
   {
   strcpy(valeur_oi.classe,   information_op_prim->information.archiver.
                              classe);
   strcpy(valeur_oi.type,     information_op_prim->information.archiver.type);
   strcpy(valeur_oi.nom,      information_op_prim->information.archiver.nom);
```

```
   strcpy(valeur_oi.etat,     information_op_prim->information.archiver.etat);
   strcpy(valeur_oi.echeance, information_op_prim->information.archiver.
                              echeance);
   strcpy(valeur_oi.ordre_clas  , "");
   strcpy(valeur_oi.etat_contenu, "");
   valeur_oi.num_copie = information_op_prim->information.archiver.num_copie;

   modifier_oi (&valeur_oi, &b_etat, &b_echeance, &b_contenu, &b_ordre, &cr,
                &ce);
   }

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_ARCHIVER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_trier (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   st_valeur_oi      valeur_oi;
   int               b_etat;
   int               b_echeance;
   int               b_contenu;
   int               b_ordre;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_TRIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * ON MET A JOUR LES ATTRIBUTS ETAT, ECHEANCE ET ORDRE_CLAS DE L'OI     *
 *****************************************************************************/

strcpy(valeur_oi.classe,information_op_prim->information.trier.classe);
strcpy(valeur_oi.type,  information_op_prim->information.trier.type);
strcpy(valeur_oi.nom,   information_op_prim->information.trier.nom);
valeur_oi.num_copie =   information_op_prim->information.trier.num_copie;
strcpy(valeur_oi.etat,  information_op_prim->information.trier.etat);
strcpy(valeur_oi.echeance,
                        information_op_prim->information.trier.echeance);
```

```
strcpy(valeur_oi.ordre_clas,information_op_prim->information.trier.ordre_tri);
strcpy(valeur_oi.etat_contenu,"VIDE");

b_etat     = VRAI;
b_echeance = VRAI;
b_contenu  = FAUX;
b_ordre    = VRAI;

if (strcmp(valeur_oi.classe,"PILE") != 0)
{  modifier_oi(&valeur_oi,&b_etat,&b_echeance,&b_contenu,&b_ordre,

&code_retour_tmp,&code_erreur_tmp);
}

/*************************************************************************
 * ON ASSOCIE CET OI A LA TABLE DE TRAVAIL                               *
 *************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.trier.classe);
strcpy(identifiant_oi.type,information_op_prim->information.trier.type);
strcpy(identifiant_oi.nom,information_op_prim->information.trier.nom);
identifiant_oi.num_copie = information_op_prim->information.trier.num_copie;

outil_travail_oi(&identifiant_oi,"TABLE_DE_TRAVAIL",&code_retour_tmp,
                                                    &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_TRIER.\n");
#endif
}

/*************************************************************************/

manipulation_bd_classer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_identifiant_oi identifiant_oi;
st_identifiant_oi identifiant_ois;
ty_code_retour    cr;
ty_code_erreur    ce;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_CLASSER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_erreur = OK;
```

```
/*****************************************************************************
 * a. SEPARER L'OBJET INFORMATIONNEL DE TOUT AUTRE OBJET : SAUF SES          *
 *    CONSTITUANTS                                                           *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.classer.classe);
strcpy(identifiant_oi.type,information_op_prim->information.classer.type);
strcpy(identifiant_oi.nom,information_op_prim->information.classer.nom);
identifiant_oi.num_copie = information_op_prim->information.classer.num_copie;

detacher_oi (&identifiant_oi, &cr, &ce);

/*****************************************************************************
 * b. ASSOCIER L'OI A CLASSER A SON OBJET DE CLASSEMENT                      *
 *****************************************************************************/

strcpy (identifiant_ois.classe, information_op_prim->information.classer.
                                 dest_classe);
strcpy (identifiant_ois.type  , information_op_prim->information.classer.
                                 dest_type);
strcpy (identifiant_ois.nom   , information_op_prim->information.classer.
                                 dest_nom);
identifiant_ois.num_copie = 0;

classer_oi (&identifiant_oi, &identifiant_ois, &cr, &ce);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_CLASSER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_ranger (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   char              dest_classe [MX_OR_CLASSE + 1];
   char              dest_nom [MX_OR_NOM + 1];
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_RANGER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif
```

```
*code_erreur = OK;

/*****************************************************************************
 * ON DETACHE L'OI                                                           *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.ranger.classe);
strcpy(identifiant_oi.type,information_op_prim->information.ranger.type);
strcpy(identifiant_oi.nom,information_op_prim->information.ranger.nom);
identifiant_oi.num_copie = information_op_prim->information.ranger.num_copie;

detacher_oi(&identifiant_oi,&code_retour_tmp,&code_erreur_tmp);

/*****************************************************************************
 * ON RANGE L'OI DANS SON OBJET DE RANGEMENT                                 *
 *****************************************************************************/

strcpy(dest_classe,information_op_prim->information.ranger.dest_classe);
strcpy(dest_nom   ,information_op_prim->information.ranger.dest_nom);

ranger_oi(&identifiant_oi,dest_classe,dest_nom,&code_retour_tmp,
                                               &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_RANGER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_replacer (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{
st_liste_identifiant_oi *liste_oi[1];
st_liste_identifiant_oi *ptr;
ty_code_retour          cr;
ty_code_erreur          ce;
int                     code_recherche;
char                    classe [MX_LG_CLASSE + 1];
char                    type [MX_LG_TYPE + 1];
st_identifiant_oi       identifiant_oi;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_REPLACER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-------------------------------------------------------\n");
#endif
```

```
*code_erreur = OK;

/*****************************************************************************
 * a. CONSTITUER LA LISTE DES OI A REPLACER                                  *
 *****************************************************************************/

liste_oi [0] = NULL;

/*****************************************************************************
 * a.1 UN OI A ETE SPECIFIE                                                  *
 *****************************************************************************/

if (strcmp (information_op_prim->information.replacer.classe, "") != 0)
   {
   liste_oi [0] = (st_liste_identifiant_oi *)
                  malloc (sizeof (st_liste_identifiant_oi));
   strcpy (liste_oi [0]->classe, information_op_prim->information.replacer.
                                 classe);
   strcpy (liste_oi [0]->type  , information_op_prim->information.replacer.
                                 type);
   strcpy (liste_oi [0]->nom   , information_op_prim->information.replacer.
                                 nom);
   liste_oi [0]->num_copie = information_op_prim->information.replacer.
                             num_copie;
   liste_oi [0]->ptr_suivant = NULL;
   }

/*****************************************************************************
 * a.2 IL FAUT REPLACER LES OI REPONDANT A UN CRITERE                        *
 *****************************************************************************/

if (strcmp (information_op_prim->information.replacer.classe, "") == 0)
   {
   if (strcmp (information_op_prim->information.replacer.gr_classe, "") != 0)
      if (strcmp (information_op_prim->information.replacer.gr_type, "") != 0)
         code_recherche = 3;
         else code_recherche = 2;
      else code_recherche = 1;

   strcpy (classe, information_op_prim->information.replacer.gr_classe);
   strcpy (type, information_op_prim->information.replacer.gr_type);

   rechercher_oi (&code_recherche, classe, type, liste_oi, &cr, &ce);
   }

/*****************************************************************************
 * b. REPLACER TOUS LES OI SE TROUVANT DANS LA LISTE                         *
 *****************************************************************************/

while (liste_oi [0] != NULL)
      {
      strcpy (identifiant_oi.classe, liste_oi [0]->classe);
      strcpy (identifiant_oi.type, liste_oi [0]->type);
      strcpy (identifiant_oi.nom, liste_oi [0]->nom);
      identifiant_oi.num_copie = liste_oi [0]->num_copie;

      replacer (&identifiant_oi, &cr, &ce);
```

```
      ptr = liste_oi [0];
      liste_oi [0] = liste_oi [0]->ptr_suivant;
      free (ptr);
      }

#if DEBUG_MANIPULATION_BD
  printf("-----------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_REPLACER.\n");
#endif
}

/*****************************************************************************/

manipulation_bd_decider (information_op_prim,code_erreur)

st_operation   *information_op_prim;
ty_code_erreur *code_erreur;

{  st_identifiant_oi identifiant_oi;
   ty_code_retour    code_retour_tmp;
   ty_code_erreur    code_erreur_tmp;

#if DEBUG_MANIPULATION_BD
  printf(">>>MANIPULATION_BD_DECIDER.\n");
  printf("-----------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Nom de l'operation  : %s\n",information_op_prim->nom_operation);
  printf("Type de l'operation : %s\n",information_op_prim->type_operation);
  printf("-----------------------------------------------------------\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * ON ASSOCIE CET OI A LA TABLE DE TRAVAIL                                   *
 *****************************************************************************/

strcpy(identifiant_oi.classe,information_op_prim->information.decider.classe);
strcpy(identifiant_oi.type,information_op_prim->information.decider.type);
strcpy(identifiant_oi.nom,information_op_prim->information.decider.nom);
identifiant_oi.num_copie = information_op_prim->information.decider.num_copie;

outil_travail_oi(&identifiant_oi,"TABLE_DE_TRAVAIL",&code_retour_tmp,
                                                    &code_erreur_tmp);

#if DEBUG_MANIPULATION_BD
  printf("-----------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------------\n");
  printf("<<<MANIPULATION_BD_DECIDER.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : msg_erreur.c                                                    */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion des messages       */
/*               d'erreur.                                                   */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"

/*****************************************************************************/

message_erreur (numero)

int numero;

{
#if DEBUG_MESSAGE_ERREUR
  printf(">>>MESSAGE_ERREUR.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Numero : %d.\n",numero);
  printf("-------------------------------------------------------\n");
#endif

  switch (numero)
  {
    case OK : break;
    default : break;
  }

#if DEBUG_MESSAGE_ERREUR
  printf("<<<MESSAGE_ERREUR.\n");
#endif
}

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

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : bd.h                                                            */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient les declarations du module de gestion */
/*               de la bd.                                                   */
/*                                                                           */
/*****************************************************************************/


/*****************************************************************************
 * STRUCTURE DES FICHIERS POUR LA BASE DE DONNEES DU LANGAGE                 *
 *****************************************************************************/

typedef struct {
              char nom_tache                          [MX_ID_TACHE + 1];
              char description                        [MX_LG_DESCRIPTION + 1];
              char nom_sous_schema_declenche          [MX_ID_SOUS_SCHEMA + 1];
              }
              st_bd_tache;

/*****************************************************************************/

typedef struct {
              char nom_sous_schema                    [MX_ID_SOUS_SCHEMA + 1];
              char description                        [MX_LG_DESCRIPTION + 1];
              int  type_concept_declenche;
              char id_concept_declenche               [MX_ID_CONCEPT + 1];
              }
              st_bd_sous_schema;

/*****************************************************************************/

typedef struct {
              char nom_boucle                         [MX_ID_BOUCLE + 1];
              char description                        [MX_LG_DESCRIPTION + 1];
              char nom_variable                       [MX_LG_VARIABLE + 1];
              char fin_sous_schema                    [MX_ID_SOUS_SCHEMA + 1];
              int  type_critere;
              int  type_concept_declenche;
              char id_concept_declenche               [MX_ID_CONCEPT + 1];
              char sous_schema_declenche              [MX_ID_SOUS_SCHEMA + 1];
              }
              st_bd_boucle;

/*****************************************************************************/

typedef struct {
              char nom_boucle_critere                 [MX_ID_BOUCLE + 1];
              union {
                    struct {
                           char classe                [MX_LG_CLASSE + 1];
                           char type                  [MX_LG_TYPE + 1];
                           char nom                   [MX_LG_NOM + 1];
                           char variable              [MX_LG_VARIABLE + 1];
                           char desir                 [MX_LG_DESIR + 1];
                           }
```

```
                        constituant;
                    struct {
                        char classe                 [MX_LG_CLASSE + 1];
                        }
                        classe;
                    }
                    type_critere;
                }
                st_bd_critere;

/*****************************************************************************/

typedef struct {
                char nom_condition                  [MX_ID_CONDITION + 1];
                char description                    [MX_LG_DESCRIPTION + 1];
                int  type_clause;
                }
                st_bd_condition;

/*****************************************************************************/

typedef struct {
                int  evaluation;
                char nom_condition                  [MX_ID_CONDITION + 1];
                char fin_sous_schema                [MX_ID_SOUS_SCHEMA + 1];
                int  type_concept_declenche;
                char id_concept_declenche           [MX_ID_CONCEPT + 1];
                }
                st_bd_cond_declenche;

/*****************************************************************************/

typedef struct {
                char nom_condition                   [MX_ID_CONDITION + 1];
                union {
                    struct {
                        char nom_attribut           [MX_LG_ATTRIBUT + 1];
                        char cst_alpha_numerique    [MX_LG_CST + 1];
                        char operateur              [MX_LG_OPERATEUR + 1];
                        char nom_variable           [MX_LG_VARIABLE + 1];
                        char classe                 [MX_LG_CLASSE + 1];
                        char type                   [MX_LG_TYPE + 1];
                        char nom                    [MX_LG_NOM + 1];
                        int  num_copie;
                        }
                        attribut;
                    struct {
                        char ois_nom_variable       [MX_LG_VARIABLE + 1];
                        char ois_classe             [MX_LG_CLASSE + 1];
                        char ois_type               [MX_LG_TYPE + 1];
                        char ois_nom                [MX_LG_NOM + 1];
                        char oi_nom_variable        [MX_LG_VARIABLE + 1];
                        char oi_classe              [MX_LG_CLASSE + 1];
                        char oi_type                [MX_LG_TYPE + 1];
                        char oi_nom                 [MX_LG_NOM + 1];
                        int  oi_num_copie;
                        }
```

```
                        appartenance;
                    struct {
                        char nom_parametre        [MX_LG_PARAMETRE + 1];
                        char operateur            [MX_LG_OPERATEUR + 1];
                        char cst_alpha_numerique [MX_LG_CST + 1];
                        }
                        parametre;
                    }
                    type_clause;
                }
                st_bd_clause;

/*****************************************************************************/

typedef struct {
                char nom_operation                [MX_ID_OPERATION + 1];
                char description                  [MX_LG_DESCRIPTION + 1];
                char fin_sous_schema              [MX_ID_SOUS_SCHEMA + 1];
                char type_operation               [MX_TY_OPERATION + 1];
                int  type_concept_declenche;
                char id_concept_declenche         [MX_ID_CONCEPT + 1];
                }
                st_bd_op_primitive;

/*****************************************************************************/

typedef struct {
                char  nom_operation               [MX_ID_OPERATION + 1];
                short constituant_reception;
                union {
                    st_creer                      creer;
                    st_reproduire_var             reproduire;
                    st_recevoir                   recevoir;
                    st_communiquer_var            communiquer;
                    st_consulter_var              consulter;
                    st_feuilleter_var             feuilleter;
                    st_enlever_var                enlever;
                    st_modifier_var               modifier;
                    st_completer_var              completer;
                    st_verifier_var               verifier;
                    st_detruire_var               detruire;
                    st_archiver_var               archiver;
                    st_trier_var                  trier;
                    st_classer_var                classer;
                    st_ranger_var                 ranger;
                    st_replacer_var               replacer;
                    st_decider_var                decider;
                    st_constituant_reception      const_reception;
                    }
                    primitive;
                }
                st_bd_operation;
```

```
/*****************************************************************************
 * STRUCTURES DE DONNEES POUR LA BASE DE DONNEES INFORMATIONNELLES           *
 *****************************************************************************/

typedef struct {
                st_identifiant_oi identifiant_oi;
                char etat                                   [MX_LG_ETAT + 1];
                char echeance                               [MX_LG_ECHEANCE + 1];
                char ordre_clas                             [MX_LG_ORDRE + 1];
                char etat_contenu                           [MX_LG_ETAT_CONTENU + 1];
                st_identifiant_oi id_original;
                char obj_int_classe                         [MX_OI_CLASSE + 1];
                char obj_int_nom                            [MX_OI_NOM + 1];
                char obj_rang_classe                        [MX_OR_CLASSE + 1];
                char obj_rang_nom                           [MX_OR_NOM + 1];
                char obj_trav_classe                        [MX_OT_CLASSE + 1];
                char obj_trav_nom                           [MX_OT_NOM + 1];
                }
                st_bd_information;

/*****************************************************************************/

typedef struct {
                int  num_clas;
                st_identifiant_oi id_oi_classe;
                st_identifiant_oi id_ois;
                }
                st_bd_classement;

/*****************************************************************************/

typedef struct {
                char classe                                 [MX_OT_CLASSE + 1];
                char nom                                    [MX_OT_NOM + 1];
                }
                st_bd_outil_trav;

/*****************************************************************************/

typedef struct {
                char classe                                 [MX_OR_CLASSE + 1];
                char nom                                    [MX_OR_NOM + 1];
                char comp_classe                            [MX_OR_CLASSE + 1];
                char comp_nom                               [MX_OR_NOM + 1];
                }
                st_bd_obj_rangement;

/*****************************************************************************/

typedef struct {
                char classe                                 [MX_OI_CLASSE + 1];
                char nom                                    [MX_OI_NOM + 1];
                }
                st_bd_obj_interface;
/*****************************************************************************
 * FIN DES DECLARATIONS                                                      *
 *****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/* FICHIER : bd.c                                                            */
/*                                                                           */
/* DESCRIPTION : Ce fichier contient le module de gestion de la bd.          */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include "debug.h"
#include "define.h"
#include "type.h"
#include "bd.h"

/*****************************************************************************
 * DECLARATION DES DESCRIPTEURS DE FICHIERS                                  *
 *****************************************************************************/

static FILE *fd_tache;
static FILE *fd_sous_schema;
static FILE *fd_boucle;
static FILE *fd_critere;
static FILE *fd_condition;
static FILE *fd_cond_decl;
static FILE *fd_clause;
static FILE *fd_operation;
static FILE *fd_op_prim;
static FILE *fd_information;
static FILE *fd_classement;
static FILE *fd_outil_trav;
static FILE *fd_obj_rang;
static FILE *fd_obj_int;

/*****************************************************************************/

ouverture_bd (code_erreur)

ty_code_erreur *code_erreur;

{
#if DEBUG_BASE_DE_DONNEES
  printf(">>>OUVERTURE_BD.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * OUVERTURE DE TOUS LES FICHIERS                                            *
 *****************************************************************************/

fd_tache = fopen ("c:/msc/tmp/bd_tache.txt", "rb");
if (fd_tache == NULL)
   *code_erreur = ERREUR_BD_OUVERTURE;

if (*code_erreur == OK)
   {
   fd_sous_schema = fopen ("c:/msc/tmp/bd_sous_schema.txt","rb");
```

```
    if (fd_sous_schema == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_boucle = fopen ("c:/msc/tmp/bd_boucle.txt", "rb");
    if (fd_boucle == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_critere = fopen ("c:/msc/tmp/bd_critere.txt", "rb");
    if (fd_critere == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_condition = fopen ("c:/msc/tmp/bd_condition.txt", "rb");
    if (fd_condition == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_cond_decl = fopen ("c:/msc/tmp/bd_cond_decl.txt", "rb");
    if (fd_cond_decl == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_clause = fopen ("c:/msc/tmp/bd_clause.txt", "rb");
    if (fd_clause == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_operation = fopen ("c:/msc/tmp/bd_operation.txt", "rb");
    if (fd_operation == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_op_prim = fopen ("c:/msc/tmp/bd_op_prim.txt", "rb");
    if (fd_op_prim == NULL)
       *code_erreur = ERREUR_BD_OUVERTURE;
    }

 if (*code_erreur == OK)
    {
    fd_information = fopen ("c:/msc/tmp/bd_information.txt", "r+b");
    if (fd_information == NULL)
```

```
      *code_erreur = ERREUR_BD_OUVERTURE;
   }

if (*code_erreur == OK)
   {
   fd_classement = fopen ("c:/msc/tmp/bd_classement.txt","r+b");
   if (fd_classement == NULL)
      *code_erreur = ERREUR_BD_OUVERTURE;
   }

if (*code_erreur == OK)
   {
   fd_outil_trav = fopen ("c:/msc/tmp/bd_outil_trav.txt", "r+b");
   if (fd_outil_trav == NULL)
      *code_erreur = ERREUR_BD_OUVERTURE;
   }

if (*code_erreur == OK)
   {
   fd_obj_rang = fopen ("c:/msc/tmp/bd_obj_rang.txt", "r+b");
   if (fd_obj_rang == NULL)
      *code_erreur = ERREUR_BD_OUVERTURE;
   }

if (*code_erreur == OK)
   {
   fd_obj_int = fopen ("c:/msc/tmp/bd_obj_int.txt", "r+b");
   if (fd_obj_int == NULL)
      *code_erreur = ERREUR_BD_OUVERTURE;
   }

#if DEBUG_BASE_DE_DONNEES
  printf("-----------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
  printf("-----------------------------------------------------------\n");
  printf("<<<OUVERTURE_BD.\n");
#endif
}

/*****************************************************************************/

fermeture_bd (code_erreur)

ty_code_erreur *code_erreur;

{
#if DEBUG_BASE_DE_DONNEES
  printf(">>>FERMETURE_BD.\n");
#endif

*code_erreur = OK;

/*****************************************************************************
 * FERMETURE DE TOUS LES FICHIERS                                            *
 *****************************************************************************/

if (fclose (fd_tache) == EOF)
```

```
   *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_sous_schema) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_boucle) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_critere) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_condition) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_cond_decl) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_clause) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_operation) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_op_prim) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_information) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_classement) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_outil_trav) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_obj_rang) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

if (*code_erreur == OK)
   if (fclose (fd_obj_int) == EOF)
      *code_erreur = ERREUR_BD_FERMETURE;

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code erreur : %d.\n",*code_erreur);
```

```
  printf("-------------------------------------------------------\n");
  printf("<<<FERMETURE_BD.\n");
#endif
}

/*****************************************************************************/

critere_boucle (identifiant_boucle,info_critere,code_retour,code_erreur)

char              identifiant_boucle [MX_ID_BOUCLE + 1];
st_critere_boucle *info_critere;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{
st_bd_boucle  boucle;
st_bd_critere critere;
short         fin;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CRITERE_BOUCLE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("identifiant boucle : %s\n", identifiant_boucle);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER BOUCLE                                                 *
 *****************************************************************************/

rewind (fd_boucle);

fin = FALSE;

while (fin == FALSE)
      {
      fread (&boucle, sizeof (boucle), 1,fd_boucle);
      if (strcmp (boucle.nom_boucle, identifiant_boucle) == 0)
         fin = TRUE;
      }

/*****************************************************************************
 * LECTURE DU FICHIER CRITERE                                                *
 *****************************************************************************/

rewind (fd_critere);

fin = FALSE;

if (boucle.type_critere == CR_TOUS)
   {
   *code_retour = TROISIEME_CRITERE;
   fin = TRUE;
   }
```

```
while (fin == FALSE)
      {
      fread (&critere, sizeof (critere),1,fd_critere);
      if (strcmp (critere.nom_boucle_critere, boucle.nom_boucle) == 0)
         fin = TRUE;
      }

/*****************************************************************************
 * TRAITEMENT DES RESULTATS                                                  *
 *****************************************************************************/

if (boucle.type_critere == CR_CONSTITUANT)
   {
   strcpy (info_critere->information.critere_constituant.classe,
           critere.type_critere.constituant.classe);
   strcpy (info_critere->information.critere_constituant.type,
           critere.type_critere.constituant.type);
   strcpy (info_critere->information.critere_constituant.nom,
           critere.type_critere.constituant.nom);
   strcpy (info_critere->information.critere_constituant.desir,
           critere.type_critere.constituant.desir);
   strcpy (info_critere->information.critere_constituant.variable,
           critere.type_critere.constituant.variable);
   }

if (boucle.type_critere == CR_CLASSE)
   strcpy (info_critere->information.critere_classe.classe,
           critere.type_critere.classe.classe);

strcpy (info_critere->nom_variable, boucle.nom_variable);
info_critere->type_critere = boucle.type_critere;

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CRITERE_BOUCLE.\n");
#endif
}

/*****************************************************************************/

clause_condition (identifiant_condition,info_clause,code_retour,code_erreur)

char                identifiant_condition [MX_ID_CONDITION + 1];
st_clause_condition *info_clause;
ty_code_retour      *code_retour;
ty_code_erreur      *code_erreur;

{
st_bd_condition      condition;
st_bd_clause         clause;
short                fin;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CLAUSE_CONDITION.\n");
```

```
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("identifiant condition : %s\n", identifiant_condition);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER CONDITION                                              *
 *****************************************************************************/

rewind (fd_condition);
fin = FALSE;

while (fin == FALSE)
      {
      fread (&condition, sizeof (condition),1,fd_condition);
      if (strcmp (condition.nom_condition, identifiant_condition) == 0)
         fin = TRUE;
      }

/*****************************************************************************
 * LECTURE DU FICHIER CLAUSE                                                 *
 *****************************************************************************/

rewind (fd_clause);
fin = FALSE;

while (fin == FALSE)
      {
      fread (&clause, sizeof (clause),1,fd_clause);
      if (strcmp (clause.nom_condition, condition.nom_condition) == 0)
         fin = TRUE;
      }

/*****************************************************************************
 * EXPLOITATION DES RESULTATS                                                *
 *****************************************************************************/

switch (condition.type_clause)
       {
       case CL_ATTRIBUT :

            strcpy (info_clause->information.clause_attribut.nom_attribut,
                    clause.type_clause.attribut.nom_attribut);
            strcpy (info_clause->information.clause_attribut.cst_alpha_num,
                    clause.type_clause.attribut.cst_alpha_numerique);
            strcpy (info_clause->information.clause_attribut.operateur,
                    clause.type_clause.attribut.operateur);
            strcpy (info_clause->information.clause_attribut.nom_variable,
                    clause.type_clause.attribut.nom_variable);
            strcpy (info_clause->information.clause_attribut.classe,
                    clause.type_clause.attribut.classe);
            strcpy (info_clause->information.clause_attribut.type,
                    clause.type_clause.attribut.type);
            strcpy (info_clause->information.clause_attribut.nom,
```

```
                    clause.type_clause.attribut.nom);
            info_clause->information.clause_attribut.num_copie =
                        clause.type_clause.attribut.num_copie;
            break;

      case CL_APPARTENANCE :

          strcpy(info_clause->information.clause_appartenance.oi_nom_variable,
                 clause.type_clause.appartenance.oi_nom_variable);
          strcpy(info_clause->information.clause_appartenance.oi_classe,
                 clause.type_clause.appartenance.oi_classe);
          strcpy(info_clause->information.clause_appartenance.oi_type,
                 clause.type_clause.appartenance.oi_type);
          strcpy(info_clause->information.clause_appartenance.oi_nom,
                 clause.type_clause.appartenance.oi_nom);
          info_clause->information.clause_appartenance.oi_num_copie =
                      clause.type_clause.appartenance.oi_num_copie;
         strcpy(info_clause->information.clause_appartenance.ois_nom_variable,
                 clause.type_clause.appartenance.ois_nom_variable);
          strcpy(info_clause->information.clause_appartenance.ois_classe,
                 clause.type_clause.appartenance.ois_classe);
          strcpy(info_clause->information.clause_appartenance.ois_type,
                 clause.type_clause.appartenance.ois_type);
          strcpy(info_clause->information.clause_appartenance.ois_nom,
                 clause.type_clause.appartenance.ois_nom);
          break;

        case CL_PARAMETRE :

            strcpy (info_clause->information.clause_parametre.nom_parametre,
                    clause.type_clause.parametre.nom_parametre);
            strcpy (info_clause->information.clause_parametre.cst_alpha_num,
                    clause.type_clause.parametre.cst_alpha_numerique);
            strcpy (info_clause->information.clause_parametre.operateur,
                    clause.type_clause.parametre.operateur);
            break;
      }

info_clause->type_clause = condition.type_clause;

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CLAUSE_CONDITION.\n");
#endif
}

/*****************************************************************************/

sous_type_operation_primitive (identifiant_operation,info_operation,
                               code_retour,code_erreur)

char             identifiant_operation [MX_ID_OPERATION + 1];
st_operation_var *info_operation;
ty_code_retour   *code_retour;
ty_code_erreur   *code_erreur;
```

```
{
short           fin;
st_bd_op_primitive op_prim;
st_bd_operation    operation;
st_liste_valeur_oi *ptr,*anc_ptr;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>SOUS_TYPE_OPERATION_PRIMITIVE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("identifiant de l'operation : %s\n", identifiant_operation);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER OP_PRIM                                                *
 *****************************************************************************/

rewind (fd_op_prim);
fin = FALSE;

while (fin == FALSE)
      {
      fread (&op_prim, sizeof (op_prim), 1,fd_op_prim);
      if (strcmp (op_prim.nom_operation, identifiant_operation) == 0)
         fin = TRUE;
      }

/*****************************************************************************
 * LECTURE DU FICHIER OPERATION                                              *
 *****************************************************************************/

rewind (fd_operation);
fin = FALSE;

while (fin == FALSE)
      {
      fread (&operation, sizeof (operation),1,fd_operation);
      if (strcmp (operation.nom_operation, op_prim.nom_operation) == 0)
         fin = TRUE;
      }

/*****************************************************************************
 * EXPLOITATION DES RESULTATS                                                *
 *****************************************************************************/

strcpy (info_operation->type_operation, op_prim.type_operation);
strcpy (info_operation->nom_operation, op_prim.nom_operation);
strcpy (info_operation->description, op_prim.description);

if (strcmp  (op_prim.type_operation, "CREER") == 0)
            {
            strcpy (info_operation->information.creer.classe,
                    operation.primitive.creer.classe);
```

```
                strcpy (info_operation->information.creer.type,
                        operation.primitive.creer.type);
                strcpy (info_operation->information.creer.nom,
                        operation.primitive.creer.nom);
                strcpy (info_operation->information.creer.etat,
                        operation.primitive.creer.etat);
                strcpy (info_operation->information.creer.echeance,
                        operation.primitive.creer.echeance);
                strcpy (info_operation->information.creer.ordre_clas,
                        operation.primitive.creer.ordre_clas);
                }

        if (strcmp  (op_prim.type_operation, "REPRODUIRE" ) == 0)
                {
                strcpy (info_operation->information.reproduire.nom_variable,
                        operation.primitive.reproduire.nom_variable);
                strcpy (info_operation->information.reproduire.classe,
                        operation.primitive.reproduire.classe);
                strcpy (info_operation->information.reproduire.type,
                        operation.primitive.reproduire.type);
                strcpy (info_operation->information.reproduire.nom,
                        operation.primitive.reproduire.nom);
                info_operation->information.reproduire.num_copie =
                                operation.primitive.reproduire.num_copie;
                info_operation->information.reproduire.nbr_copie =
                                operation.primitive.reproduire.nbr_copie;
                info_operation->information.reproduire.num_depart =
                                operation.primitive.reproduire.num_depart;
                }

        if (strcmp  (op_prim.type_operation, "RECEVOIR" ) == 0)
                {
                strcpy (info_operation->information.recevoir.classe,
                        operation.primitive.recevoir.classe);
                strcpy (info_operation->information.recevoir.type,
                        operation.primitive.recevoir.type);
                strcpy (info_operation->information.recevoir.nom,
                        operation.primitive.recevoir.nom);
                strcpy (info_operation->information.recevoir.etat,
                        operation.primitive.recevoir.etat);
                strcpy (info_operation->information.recevoir.echeance,
                        operation.primitive.recevoir.echeance);
                strcpy (info_operation->information.recevoir.ordre_clas,
                        operation.primitive.recevoir.ordre_clas);
                strcpy (info_operation->information.recevoir.mode_reception,
                        operation.primitive.recevoir.mode_reception);
                info_operation->information.recevoir.num_copie =
                                operation.primitive.recevoir.num_copie;
                info_operation->information.recevoir.composants = NULL;

                fin     = FALSE;
                anc_ptr = NULL;

                rewind (fd_operation);

                if (fread (&operation, sizeof (operation), 1, fd_operation) == 0)
                   fin = TRUE;
```

```
                while (fin == FALSE)
                      {
                      if (operation.constituant_reception == VRAI)
                         if (strcmp (operation.primitive.const_reception.
                                     nom_operation, identifiant_operation) == 0)
                            {
                            ptr = (st_liste_valeur_oi *)
                                  malloc (sizeof (st_liste_valeur_oi));
                            strcpy (ptr->classe, operation.primitive.
                                  const_reception.identifiant_oi.classe);
                            strcpy (ptr->type, operation.primitive.
                                  const_reception.identifiant_oi.type);
                            strcpy (ptr->nom, operation.primitive.
                                  const_reception.identifiant_oi.nom);
                            strcpy (ptr->etat, operation.primitive.
                                  const_reception.etat);
                            strcpy (ptr->echeance, operation.primitive.
                                  const_reception.echeance);
                            ptr->num_copie = operation.primitive.const_reception.
                                             identifiant_oi.num_copie;
                            ptr->ptr_suivant = NULL;
                            if (info_operation->information.recevoir.composants
                               == NULL)
                              info_operation->information.recevoir.composants=ptr;
                              else anc_ptr->ptr_suivant = ptr;
                            anc_ptr = ptr;
                            }
                      if (fread (&operation, sizeof (operation), 1,
                         fd_operation) == 0)
                         fin = TRUE;
                      }
                }

if (strcmp  (op_prim.type_operation, "COMMUNIQUER" ) == 0)
                {
                strcpy(info_operation->information.communiquer.nom_variable,
                       operation.primitive.communiquer.nom_variable);
                strcpy(info_operation->information.communiquer.classe,
                       operation.primitive.communiquer.classe);
                strcpy(info_operation->information.communiquer.type,
                       operation.primitive.communiquer.type);
                strcpy(info_operation->information.communiquer.nom,
                       operation.primitive.communiquer.nom);
                strcpy(info_operation->information.communiquer.etat,
                       operation.primitive.communiquer.etat);
                strcpy(info_operation->information.communiquer.echeance,
                       operation.primitive.communiquer.echeance);
                strcpy(info_operation->information.communiquer.mode_communication,
                       operation.primitive.communiquer.mode_communication);
                info_operation->information.communiquer.num_copie =
                               operation.primitive.communiquer.num_copie;
                }

if (strcmp  (op_prim.type_operation, "CONSULTER" ) == 0)
                {
                strcpy (info_operation->information.consulter.nom_variable,
```

```
                operation.primitive.consulter.nom_variable);
        strcpy (info_operation->information.consulter.classe,
                operation.primitive.consulter.classe);
        strcpy (info_operation->information.consulter.type,
                operation.primitive.consulter.type);
        strcpy (info_operation->information.consulter.nom,
                operation.primitive.consulter.nom);
        strcpy (info_operation->information.consulter.etat,
                operation.primitive.consulter.etat);
        strcpy (info_operation->information.consulter.echeance,
                operation.primitive.consulter.echeance);
        info_operation->information.consulter.num_copie =
                        operation.primitive.consulter.num_copie;
        }

if (strcmp  (op_prim.type_operation, "FEUILLETER" ) == 0)
        {
        strcpy (info_operation->information.feuilleter.nom_variable,
                operation.primitive.feuilleter.nom_variable);
        strcpy (info_operation->information.feuilleter.classe,
                operation.primitive.feuilleter.classe);
        strcpy (info_operation->information.feuilleter.type,
                operation.primitive.feuilleter.type);
        strcpy (info_operation->information.feuilleter.nom,
                operation.primitive.feuilleter.nom);
        strcpy (info_operation->information.feuilleter.etat,
                operation.primitive.feuilleter.etat);
        strcpy (info_operation->information.feuilleter.echeance,
                operation.primitive.feuilleter.echeance);
        strcpy (info_operation->information.feuilleter.ordre_feuilletage,
                operation.primitive.feuilleter.ordre_feuilletage);
        }

if (strcmp  (op_prim.type_operation, "ENLEVER" ) == 0)
        {
        strcpy (info_operation->information.enlever.nom_variable,
                operation.primitive.enlever.nom_variable);
        strcpy (info_operation->information.enlever.nom_pile,
                operation.primitive.enlever.nom_pile);
        strcpy (info_operation->information.enlever.variable_constituant,
                operation.primitive.enlever.variable_constituant);
        }

if (strcmp  (op_prim.type_operation, "MODIFIER" ) == 0)
        {
        strcpy (info_operation->information.modifier.nom_variable,
                operation.primitive.modifier.nom_variable);
        strcpy (info_operation->information.modifier.type,
                operation.primitive.modifier.type);
        strcpy (info_operation->information.modifier.nom,
                operation.primitive.modifier.nom);
        strcpy (info_operation->information.modifier.etat,
                operation.primitive.modifier.etat);
        strcpy (info_operation->information.modifier.echeance,
                operation.primitive.modifier.echeance);
        strcpy (info_operation->information.modifier.nom_message,
                operation.primitive.modifier.nom_message);
```

```
                    info_operation->information.modifier.num_copie =
                                operation.primitive.modifier.num_copie;
            info_operation->information.modifier.num_copie_message =
                            operation.primitive.modifier.num_copie_message;
            }

if (strcmp  (op_prim.type_operation, "COMPLETER" ) == 0)
            {
            strcpy (info_operation->information.completer.nom_variable,
                    operation.primitive.completer.nom_variable);
            strcpy (info_operation->information.completer.type,
                    operation.primitive.completer.type);
            strcpy (info_operation->information.completer.nom,
                    operation.primitive.completer.nom);
            strcpy (info_operation->information.completer.echeance,
                    operation.primitive.completer.echeance);
            strcpy (info_operation->information.completer.etat,
                    operation.primitive.completer.etat);
            info_operation->information.completer.num_copie =
                            operation.primitive.completer.num_copie;
            }

if (strcmp  (op_prim.type_operation, "VERIFIER" ) == 0)
            {
            strcpy (info_operation->information.verifier.nom_variable,
                    operation.primitive.verifier.nom_variable);
            strcpy (info_operation->information.verifier.classe,
                    operation.primitive.verifier.classe);
            strcpy (info_operation->information.verifier.type,
                    operation.primitive.verifier.type);
            strcpy (info_operation->information.verifier.nom,
                    operation.primitive.verifier.nom);
            strcpy (info_operation->information.verifier.etat,
                    operation.primitive.verifier.etat);
            strcpy (info_operation->information.verifier.echeance,
                    operation.primitive.verifier.echeance);
            strcpy (info_operation->information.verifier.nom_parametre,
                    operation.primitive.verifier.nom_parametre);
            info_operation->information.verifier.num_copie =
                            operation.primitive.verifier.num_copie;
            }

if (strcmp  (op_prim.type_operation, "DETRUIRE" ) == 0)
            {
            strcpy (info_operation->information.detruire.nom_variable,
                    operation.primitive.detruire.nom_variable);
            strcpy (info_operation->information.verifier.classe,
                    operation.primitive.verifier.classe);
            strcpy (info_operation->information.verifier.type,
                    operation.primitive.verifier.type);
            strcpy (info_operation->information.verifier.nom,
                    operation.primitive.verifier.nom);
            strcpy (info_operation->information.verifier.etat,
                    operation.primitive.verifier.etat);
            strcpy (info_operation->information.verifier.echeance,
                    operation.primitive.verifier.echeance);
            info_operation->information.detruire.num_copie =
```

```
                        operation.primitive.detruire.num_copie;
                }

        if (strcmp  (op_prim.type_operation, "ARCHIVER" ) == 0)
                {
                strcpy (info_operation->information.archiver.nom_variable,
                        operation.primitive.archiver.nom_variable);
                strcpy (info_operation->information.archiver.classe,
                        operation.primitive.archiver.classe);
                strcpy (info_operation->information.archiver.type,
                        operation.primitive.archiver.type);
                strcpy (info_operation->information.archiver.nom,
                        operation.primitive.archiver.nom);
                strcpy (info_operation->information.archiver.etat,
                        operation.primitive.archiver.etat);
                strcpy (info_operation->information.archiver.echeance,
                        operation.primitive.archiver.echeance);
                info_operation->information.archiver.num_copie =
                                operation.primitive.archiver.num_copie;
                }

        if (strcmp  (op_prim.type_operation, "TRIER" ) == 0)
                {
                strcpy (info_operation->information.trier.nom_variable,
                        operation.primitive.trier.nom_variable);
                strcpy (info_operation->information.trier.classe,
                        operation.primitive.trier.classe);
                strcpy (info_operation->information.trier.type,
                        operation.primitive.trier.type);
                strcpy (info_operation->information.trier.nom,
                        operation.primitive.trier.nom);
                strcpy (info_operation->information.trier.etat,
                        operation.primitive.trier.etat);
                strcpy (info_operation->information.trier.echeance,
                        operation.primitive.trier.echeance);
                strcpy (info_operation->information.trier.ordre_tri,
                        operation.primitive.trier.ordre_tri);
                info_operation->information.trier.num_copie =
                                operation.primitive.trier.num_copie;
                }

        if (strcmp  (op_prim.type_operation, "CLASSER" ) == 0)
                {
                strcpy (info_operation->information.classer.nom_variable,
                        operation.primitive.classer.nom_variable);
                strcpy (info_operation->information.classer.classe,
                        operation.primitive.classer.classe);
                strcpy (info_operation->information.classer.type,
                        operation.primitive.classer.type);
                strcpy (info_operation->information.classer.nom,
                        operation.primitive.classer.nom);
                strcpy (info_operation->information.classer.dest_classe,
                        operation.primitive.classer.dest_classe);
                strcpy (info_operation->information.classer.dest_type,
                        operation.primitive.classer.dest_type);
                strcpy (info_operation->information.classer.dest_nom,
                        operation.primitive.classer.dest_nom);
```

```
                info_operation->information.classer.num_copie =
                                operation.primitive.classer.num_copie;
                }

        if (strcmp  (op_prim.type_operation, "RANGER" ) == 0)
                {
                strcpy (info_operation->information.ranger.nom_variable,
                        operation.primitive.ranger.nom_variable);
                strcpy (info_operation->information.ranger.classe,
                        operation.primitive.ranger.classe);
                strcpy (info_operation->information.ranger.type,
                        operation.primitive.ranger.type);
                strcpy (info_operation->information.ranger.nom,
                        operation.primitive.ranger.nom);
                strcpy (info_operation->information.ranger.dest_classe,
                        operation.primitive.ranger.dest_classe);
                strcpy (info_operation->information.ranger.dest_nom,
                        operation.primitive.ranger.dest_nom);
                info_operation->information.ranger.num_copie =
                                operation.primitive.ranger.num_copie;
                }

        if (strcmp  (op_prim.type_operation, "REPLACER" ) == 0)
                {
                strcpy (info_operation->information.replacer.nom_variable,
                        operation.primitive.replacer.nom_variable);
                strcpy (info_operation->information.replacer.classe,
                        operation.primitive.replacer.classe);
                strcpy (info_operation->information.replacer.type,
                        operation.primitive.replacer.type);
                strcpy (info_operation->information.replacer.nom,
                        operation.primitive.replacer.nom);
                strcpy (info_operation->information.replacer.gr_classe,
                        operation.primitive.replacer.gr_classe);
                strcpy (info_operation->information.replacer.gr_type,
                        operation.primitive.replacer.gr_type);
                info_operation->information.replacer.num_copie =
                                operation.primitive.replacer.num_copie;
                }

        if (strcmp  (op_prim.type_operation, "DECIDER" ) == 0)
                {
                strcpy (info_operation->information.decider.nom_variable,
                        operation.primitive.decider.nom_variable);
                strcpy (info_operation->information.decider.classe,
                        operation.primitive.decider.classe);
                strcpy (info_operation->information.decider.type,
                        operation.primitive.decider.type);
                strcpy (info_operation->information.decider.nom,
                        operation.primitive.decider.nom);
                strcpy (info_operation->information.decider.nom_parametre,
                        operation.primitive.decider.nom_parametre);
                info_operation->information.decider.num_copie =
                                operation.primitive.decider.num_copie;
                }
```

```
#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  printf("type_operation declenchee : %s\n", info_operation->type_operation);
  printf("nom de l'operation : %s\n", info_operation->nom_operation);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SOUS_TYPE_OPERATION_PRIMITIVE.\n");
#endif
}

/*****************************************************************************/

tache_concept_suivant (identifiant_tache,concept_declenche,code_retour,
                       code_erreur)

char           identifiant_tache [MX_ID_TACHE + 1];
st_c_declenche *concept_declenche;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{ st_bd_tache tache;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>TACHE_CONCEPT_SUIVANT.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DU CONCEPT SUIVANT DE LA TACHE                                  *
 *****************************************************************************/

rewind (fd_tache);

do
{  fread (&tache,sizeof(tache),1,fd_tache);
}
while (strcmp(identifiant_tache,tache.nom_tache) != 0);

concept_declenche->type_concept = SOUS_SCHEMA;
strcpy (concept_declenche->id_concept_declenche,
                                        tache.nom_sous_schema_declenche);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Concept suivant declenche :\n");
  printf("Type : %d.\n",concept_declenche->type_concept);
  printf("Nom  : %s.\n",concept_declenche->id_concept_declenche);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
```

```
  printf("<<<TACHE_CONCEPT_SUIVANT.\n");
#endif
}

/*****************************************************************************/

sous_schema_concept_suivant (identifiant_sous_schema,concept_declenche,
                             code_retour,code_erreur)

char           identifiant_sous_schema [MX_ID_SOUS_SCHEMA + 1];
st_c_declenche *concept_declenche;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_bd_sous_schema sous_schema;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>SOUS_SCHEMA_CONCEPT_SUIVANT.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DU CONCEPT SUIVANT DU SOUS-SCHEMA                               *
 *****************************************************************************/

rewind (fd_sous_schema);

do
{  fread (&sous_schema,sizeof(sous_schema),1,fd_sous_schema);
}
while (strcmp(identifiant_sous_schema,sous_schema.nom_sous_schema) != 0);

concept_declenche->type_concept = sous_schema.type_concept_declenche;
strcpy (concept_declenche->id_concept_declenche,
                                        sous_schema.id_concept_declenche);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Concept suivant declenche :\n");
  printf("Type : %d.\n",concept_declenche->type_concept);
  printf("Nom  : %s.\n",concept_declenche->id_concept_declenche);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<SOUS_SCHEMA_CONCEPT_SUIVANT.\n");
#endif
}

/*****************************************************************************/

boucle_concept_suivant (identifiant_boucle,critere,concept_declenche,
                        code_retour,code_erreur)
```

```
char           identifiant_boucle [MX_ID_BOUCLE + 1];
int            *critere;
st_c_declenche *concept_declenche;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_bd_boucle boucle;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>BOUCLE_CONCEPT_SUIVANT.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DU CONCEPT SUIVANT DE LA BOUCLE                                 *
 *****************************************************************************/

rewind (fd_boucle);

do
{  fread (&boucle,sizeof(boucle),1,fd_boucle);
}
while (strcmp(identifiant_boucle,boucle.nom_boucle) != 0);

if (*critere == SOUS_SCHEMA)
{  concept_declenche->type_concept = SOUS_SCHEMA;
   strcpy(concept_declenche->id_concept_declenche,
                                        boucle.sous_schema_declenche);
}
else
{  if (boucle.fin_sous_schema[0] != '\0')
   {  strcpy(concept_declenche->id_concept_declenche,boucle.fin_sous_schema);
      *code_retour = FIN_SOUS_SCHEMA;
   }
   else
   {  concept_declenche->type_concept = boucle.type_concept_declenche;
      strcpy(concept_declenche->id_concept_declenche,
                                          boucle.id_concept_declenche);
   }
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Concept suivant declenche :\n");
  printf("Type : %d.\n",concept_declenche->type_concept);
  printf("Nom  : %s.\n",concept_declenche->id_concept_declenche);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<BOUCLE_CONCEPT_SUIVANT.\n");
#endif
```

```
}

/*****************************************************************************/

condition_concept_suivant (identifiant_condition,critere,concept_declenche,
                           code_retour,code_erreur)

char           identifiant_condition [MX_ID_CONDITION + 1];
int            *critere;
st_c_declenche *concept_declenche;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_bd_cond_declenche cond_decl;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CONDITION_CONCEPT_SUIVANT.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("-------------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DU CONCEPT SUIVANT DE LA CONDITION                              *
 *****************************************************************************/

rewind (fd_cond_decl);

do
{  fread (&cond_decl,sizeof(cond_decl),1,fd_cond_decl);
}
while ((strcmp(identifiant_condition,cond_decl.nom_condition) != 0) OR
       (*critere != cond_decl.evaluation));

if (cond_decl.fin_sous_schema[0] != '\0')
{  strcpy(concept_declenche->id_concept_declenche,cond_decl.fin_sous_schema);
   *code_retour = FIN_SOUS_SCHEMA;
}
else
{  concept_declenche->type_concept = cond_decl.type_concept_declenche;
   strcpy(concept_declenche->id_concept_declenche,
                                        cond_decl.id_concept_declenche);
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Concept suivant declenche :\n");
  printf("Type : %d.\n",concept_declenche->type_concept);
  printf("Nom  : %s.\n",concept_declenche->id_concept_declenche);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONDITION_CONCEPT_SUIVANT.\n");
#endif
```

```
}

/*****************************************************************************/

operation_concept_suivant (identifiant_op_prim,concept_declenche,code_retour,
                           code_erreur)

char           identifiant_op_prim [MX_ID_OPERATION + 1];
st_c_declenche *concept_declenche;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_bd_op_primitive op_prim;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>OPERATION_CONCEPT_SUIVANT.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant op prim : %s.\n",identifiant_op_prim);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DU CONCEPT SUIVANT DE L'OPERATION PRIMITIVE                     *
 *****************************************************************************/

rewind (fd_op_prim);

do
{  fread (&op_prim,sizeof(op_prim),1,fd_op_prim);
}
while (strcmp(identifiant_op_prim,op_prim.nom_operation) != 0);

if (op_prim.fin_sous_schema[0] != '\0')
{  strcpy(concept_declenche->id_concept_declenche,op_prim.fin_sous_schema);
   *code_retour = FIN_SOUS_SCHEMA;
}
else
{  concept_declenche->type_concept = op_prim.type_concept_declenche;
   strcpy(concept_declenche->id_concept_declenche,
                                       op_prim.id_concept_declenche);
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Concept suivant declenche :\n");
  printf("Type : %d.\n",concept_declenche->type_concept);
  printf("Nom  : %s.\n",concept_declenche->id_concept_declenche);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OPERATION_CONCEPT_SUIVANT.\n");
#endif
}
```

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

variable_boucle (identifiant_boucle,variable_boucle,code_retour,code_erreur)

char           identifiant_boucle [MX_ID_BOUCLE + 1];
char           variable_boucle [MX_LG_VARIABLE + 1];
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{
st_bd_boucle boucle;
short        fin;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>VARIABLE_BOUCLE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant boucle : %s.\n",identifiant_boucle);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DE LA VARIABLE ASSOCIEE A LA BOUCLE                             *
 *****************************************************************************/

rewind (fd_boucle);

fin = FALSE;

while (fin == FALSE)
      {
      fread (&boucle, sizeof (boucle), 1, fd_boucle);
      if (strcmp (boucle.nom_boucle, identifiant_boucle) == 0)
         {
         strcpy (variable_boucle, boucle.nom_variable);
         fin = TRUE;
         }
      }

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Variable de la boucle : %s.\n",variable_boucle);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<VARIABLE_BOUCLE.\n");
#endif
}

/*****************************************************************************/

constituant_ois_tout(identifiant_ois,type_oi,liste_oi,code_retour,code_erreur)

st_identifiant_oi       *identifiant_ois;
```

```
char                    *type_oi;
st_liste_identifiant_oi *liste_oi[];
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;

{
st_liste_identifiant_oi *ptr_courant, *liste_aux [1], *ptr;
st_identifiant_oi       id_ois_classe;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CONSTITUANT_OIS_TOUT.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("identifiant_ois : %s\n", identifiant_ois);
  printf("type du critere : %s\n", type_oi);
  printf("-------------------------------------------------------\n");
#endif

liste_oi [0] = NULL;
*code_retour = *code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DE TOUS LES OI SE TROUVANT DANS L'OIS SPECIFIE A TOUS LES  *
 * NIVEAUX .                                                           *
 *****************************************************************************/

oi_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur);

ptr_courant = liste_oi [0];

while (ptr_courant != NULL AND *code_retour == OK AND *code_erreur == OK)
      {
      if (strcmp (ptr_courant->classe, "DOSSIER") == 0 OR
          strcmp (ptr_courant->classe, "FICHIER") == 0 OR
          strcmp (ptr_courant->classe, "PILE") == 0)
         {
         strcpy (id_ois_classe.classe, ptr_courant->classe);
         strcpy (id_ois_classe.type,   ptr_courant->type);
         strcpy (id_ois_classe.nom,    ptr_courant->nom);
         id_ois_classe.num_copie = ptr_courant->num_copie;
         liste_aux [0] = NULL;

         oi_dans_ois (&id_ois_classe, liste_aux, code_retour, code_erreur);

         if (liste_aux [0] != NULL)
            {
            ptr = ptr_courant->ptr_suivant;
            ptr_courant->ptr_suivant = liste_aux [0];

            while (liste_aux [0]->ptr_suivant != NULL)
                  liste_aux [0] = liste_aux [0]->ptr_suivant;

            liste_aux [0]->ptr_suivant = ptr;
            }
         }
      ptr_courant = ptr_courant->ptr_suivant;
      }
```

```
/*****************************************************************************
 * DANS LA LISTE CONSTITUEE, ON NE RETIENT QUE CEUX QUI REPONDENT AU    *
 * CRITERE SPECIFIE                                                     *
 *****************************************************************************/

/*****************************************************************************
 * RECHERCHE DE TOUS LES OIE SE TROUVANT DANS L'OIS SPECIFIE            *
 *****************************************************************************/

if (strcmp (type_oi, "OIE") == 0 AND *code_retour == OK AND *code_erreur ==
OK)
   {
   ptr_courant = liste_oi [0];
   while (ptr_courant != NULL)
         {
         if (strcmp (ptr_courant->classe, "PILE")    == 0 OR
             strcmp (ptr_courant->classe, "FICHIER") == 0 OR
             strcmp (ptr_courant->classe, "DOSSIER") == 0 )

            if (ptr_courant == liste_oi [0])
               liste_oi [0] = ptr_courant->ptr_suivant;
               else ptr->ptr_suivant = ptr_courant->ptr_suivant;
            else ptr = ptr_courant;

         ptr_courant = ptr_courant->ptr_suivant;
         }
   }

/*****************************************************************************
 * RECHERCHE DE TOUS LES OIS SE TROUVANT DANS L'OIS SPECIFIE            *
 *****************************************************************************/

if (strcmp (type_oi, "OIS") == 0 AND *code_retour == OK AND *code_erreur ==
OK)
   {
   ptr_courant = liste_oi [0];
   while (ptr_courant != NULL)
         {
         if (strcmp (ptr_courant->classe, "MESSAGE")    == 0 OR
             strcmp (ptr_courant->classe, "FORMULAIRE") == 0 OR
             strcmp (ptr_courant->classe, "DOCUMENT")   == 0 )

            if (ptr_courant == liste_oi [0])
               liste_oi [0] = ptr_courant->ptr_suivant;
               else ptr->ptr_suivant = ptr_courant->ptr_suivant;
                                                  else ptr = ptr_courant;

         ptr_courant = ptr_courant->ptr_suivant;
         }
   }

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  ptr = liste_oi[0];
  while (ptr != NULL)
```

```
        {
        printf("classe    : %s\n", ptr->classe);
        printf("type      : %s\n", ptr->type);
        printf("nom       : %s\n", ptr->nom);
        printf("num copie : %d\n\n", ptr->num_copie);
        ptr = ptr->ptr_suivant;
        }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CONSTITUANT_OIS_TOUT.\n");
#endif
}

/*****************************************************************************/

constituant_ois (identifiant_ois, type_oi, liste_oi, code_retour, code_erreur)

st_identifiant_oi       *identifiant_ois;
char                    *type_oi;
st_liste_identifiant_oi *liste_oi [];
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;

{
short                   fin_fichier;
short                   trouve;
st_liste_identifiant_oi *ptr;
st_bd_information       information;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CONSTITUANT_OIS.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("identifiant_ois : %s\n", identifiant_ois);
  printf("type du critere : %s\n", type_oi);
  printf("-------------------------------------------------------\n");
#endif

liste_oi [0] = NULL;
*code_retour = *code_erreur = OK;

/*****************************************************************************
 * REGARDER SI L'OIS EXISTE DANS LE SYSTEME                                  *
 *****************************************************************************/

fin_fichier = FALSE;
trouve      = FALSE;
rewind (fd_information);

if (fread (&information, sizeof (information), 1, fd_information) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE AND trouve == FALSE)
      {
      if (strcmp(identifiant_ois->classe, information.identifiant_oi.classe)
          == 0 AND
```

```
            strcmp(identifiant_ois->type, information.identifiant_oi.type)
            == 0 AND
            strcmp(identifiant_ois->nom, information.identifiant_oi.nom) == 0)

            trouve = TRUE;

         if (fread (&information, sizeof (information), 1, fd_information) == 0)
            fin_fichier = TRUE;
         }

   if (trouve == FALSE)
      *code_erreur = EXISTE_PAS;

   /*************************************************************************
    * EVALUATION DU CRITERE DE RECHERCHE ET TRAITEMENT EN CONSEQUENCE       *
    *************************************************************************/

   if (strcmp (type_oi, "OIE") == 0 AND trouve == TRUE)
      oie_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur);

   if (strcmp (type_oi, "OIS") == 0 AND trouve == TRUE)
      ois_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur);

   if (strcmp (type_oi, "TOUS") == 0 AND trouve == TRUE)
      oi_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur);

   #if DEBUG_BASE_DE_DONNEES
     printf("-------------------------------------------------------\n");
     printf("Resultats :\n");
     ptr = liste_oi[0];
     while (ptr != NULL)
           {
           printf("classe    : %s\n", ptr->classe);
           printf("type      : %s\n", ptr->type);
           printf("nom       : %s\n", ptr->nom);
           printf("num copie : %d\n\n", ptr->num_copie);
           ptr = ptr->ptr_suivant;
           }
     printf("-------------------------------------------------------\n");
     printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
     printf("-------------------------------------------------------\n");
     printf("<<<CONSTITUANT_OIS.\n");
   #endif
   }

   /*************************************************************************
    * RECHERCHE DE TOUS LES OIE SE TROUVANT DANS L'OIS SPECIFIE             *
    * AU PREMIER NIVEAU.                                                    *
    *************************************************************************/

   oie_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur)

   st_identifiant_oi       *identifiant_ois;
   st_liste_identifiant_oi *liste_oi[];
   ty_code_retour          *code_retour;
   ty_code_erreur          *code_erreur;
```

```
{
st_bd_classement        classement;
short                   fin_fichier;
st_liste_identifiant_oi *ptr, *anc_ptr;

fin_fichier = FALSE;
anc_ptr     = NULL;

rewind (fd_classement);

if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      if (strcmp (classement.id_ois.classe, identifiant_ois->classe) == 0 AND
          strcmp (classement.id_ois.type, identifiant_ois->type) == 0 AND
          strcmp (classement.id_ois.nom, identifiant_ois->nom) == 0 AND
          classement.id_ois.num_copie == identifiant_ois->num_copie)

          if (strcmp (classement.id_oi_classe.classe, "FORMULAIRE") == 0 OR
              strcmp (classement.id_oi_classe.classe, "MESSAGE") == 0 OR
              strcmp (classement.id_oi_classe.classe, "DOCUMENT") == 0)
              {
              ptr = (st_liste_identifiant_oi *)
                    malloc (sizeof(st_liste_identifiant_oi));
              strcpy (ptr->classe, classement.id_oi_classe.classe);
              strcpy (ptr->type, classement.id_oi_classe.type);
              strcpy (ptr->nom, classement.id_oi_classe.nom);
              ptr->num_copie = classement.id_oi_classe.num_copie;
              ptr->ptr_suivant = NULL;

              if (liste_oi [0] == NULL)
                 liste_oi [0] = ptr;
                 else anc_ptr->ptr_suivant = ptr;
              anc_ptr = ptr;
              }

      if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
         fin_fichier = TRUE;
      }
if (liste_oi [0] == NULL)
   *code_retour = VIDE;
}

/*****************************************************************************
 * RECHERCHE DE TOUS LES OIS SE TROUVANT DANS L'OIS SPECIFIE AU PREMIER *
 * NIVEAU .                                                                  *
 *****************************************************************************/

ois_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur)

st_identifiant_oi       *identifiant_ois;
st_liste_identifiant_oi *liste_oi[];
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;
```

```
{
st_bd_classement        classement;
short                   fin_fichier;
st_liste_identifiant_oi *ptr, *anc_ptr;

fin_fichier = FALSE;
anc_ptr     = NULL;

rewind (fd_classement);

if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      if (strcmp (classement.id_ois.classe, identifiant_ois->classe) == 0 AND
          strcmp (classement.id_ois.type, identifiant_ois->type) == 0 AND
          strcmp (classement.id_ois.nom, identifiant_ois->nom) == 0 AND
          classement.id_ois.num_copie == identifiant_ois->num_copie)

          if (strcmp (classement.id_oi_classe.classe, "FICHIER") == 0 OR
              strcmp (classement.id_oi_classe.classe, "DOSSIER") == 0 OR
              strcmp (classement.id_oi_classe.classe, "PILE") == 0)
              {
              ptr = (st_liste_identifiant_oi *)
                    malloc (sizeof(st_liste_identifiant_oi));
              strcpy (ptr->classe, classement.id_oi_classe.classe);
              strcpy (ptr->type, classement.id_oi_classe.type);
              strcpy (ptr->nom, classement.id_oi_classe.nom);
              ptr->num_copie = classement.id_oi_classe.num_copie;
              ptr->ptr_suivant = NULL;

              if (liste_oi [0] == NULL)
                 liste_oi [0] = ptr;
                 else anc_ptr->ptr_suivant = ptr;
              anc_ptr = ptr;
              }

      if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
         fin_fichier = TRUE;
      }
if (liste_oi [0] == NULL)
   *code_retour = VIDE;
}

/*************************************************************************
 * RECHERCHE DE TOUS LES OI SE TROUVANT DANS L'OIS SPECIFIE AU PREMIER  *
 * NIVEAU.                                                              *
 *************************************************************************/

oi_dans_ois (identifiant_ois, liste_oi, code_retour, code_erreur)

st_identifiant_oi       *identifiant_ois;
st_liste_identifiant_oi *liste_oi[];
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;
```

```
{
st_bd_classement        classement;
short                   fin_fichier;
st_liste_identifiant_oi *ptr, *anc_ptr;

fin_fichier = FALSE;
anc_ptr     = NULL;

rewind (fd_classement);

if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      if (strcmp (classement.id_ois.classe, identifiant_ois->classe) == 0 AND
          strcmp (classement.id_ois.type, identifiant_ois->type) == 0 AND
          strcmp (classement.id_ois.nom, identifiant_ois->nom) == 0 AND
          classement.id_ois.num_copie == identifiant_ois->num_copie)

          {
          ptr = (st_liste_identifiant_oi *)
                malloc (sizeof(st_liste_identifiant_oi));
          strcpy (ptr->classe, classement.id_oi_classe.classe);
          strcpy (ptr->type, classement.id_oi_classe.type);
          strcpy (ptr->nom, classement.id_oi_classe.nom);
          ptr->num_copie = classement.id_oi_classe.num_copie;
          ptr->ptr_suivant = NULL;

          if (liste_oi [0] == NULL)
             liste_oi [0] = ptr;
             else anc_ptr->ptr_suivant = ptr;
          anc_ptr = ptr;
          }

      if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
         fin_fichier = TRUE;
      }
if (liste_oi [0] == NULL)
   *code_retour = VIDE;
}

/*****************************************************************************/

objets_classe (classe,liste_oi,code_retour,code_erreur)

char                    classe [MX_LG_CLASSE + 1];
st_liste_identifiant_oi *liste_oi[];
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;

{
st_bd_information       information;
short                   fin_fichier;
st_liste_identifiant_oi *ptr, *anc_ptr;

#if DEBUG_BASE_DE_DONNEES
```

```
  printf(">>>OBJETS_CLASSE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("classe : %s\n", classe);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER INFORMATION                                            *
 *****************************************************************************/

fin_fichier = FALSE;
liste_oi[0] = NULL;
anc_ptr     = NULL;

rewind (fd_information);

if (fread (&information, sizeof (information), 1, fd_information) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      if (strcmp (information.identifiant_oi.classe, classe) == 0)
         {
         ptr = (st_liste_identifiant_oi *)
               malloc (sizeof (st_liste_identifiant_oi));
         strcpy (ptr->classe, information.identifiant_oi.classe);
         strcpy (ptr->type, information.identifiant_oi.type);
         strcpy (ptr->nom, information.identifiant_oi.nom);
         ptr->num_copie = information.identifiant_oi.num_copie;
         ptr->ptr_suivant = NULL;

         if (liste_oi[0] == NULL)
            liste_oi[0] = ptr;
            else anc_ptr->ptr_suivant = ptr;
         anc_ptr = ptr;
         }

       if (fread (&information, sizeof (information), 1, fd_information) == 0)
          fin_fichier = TRUE;

       }

if (liste_oi [0] == NULL)
   *code_retour = VIDE;

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  ptr = liste_oi[0];
  while (ptr != NULL)
        {
        printf("classe : %s\n", ptr->classe);
        printf("type : %s\n", ptr->type);
        printf("nom : %s\n", ptr->nom);
```

```
        printf("num copie : %d\n", ptr->num_copie);
        ptr = ptr->ptr_suivant;
        }
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OBJETS_CLASSE.\n");
#endif
}

/*****************************************************************************/

tous_objets (liste_oi,code_retour,code_erreur)

st_liste_identifiant_oi **liste_oi;
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;

{
short                   fin_fichier;
st_bd_information       information;
st_liste_identifiant_oi *ptr, *anc_ptr;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>TOUS_OBJETS.\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DE TOUS LES OI DU FICHIER INFORMATION                             *
 *****************************************************************************/

fin_fichier  = FALSE;
anc_ptr      = NULL;
liste_oi [0] = NULL;

rewind (fd_information);

if (fread (information, sizeof (information), 1, fd_information) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      ptr = (st_liste_identifiant_oi *)
            malloc (sizeof (st_liste_identifiant_oi));
      strcpy (ptr->classe, information.identifiant_oi.classe);
      strcpy (ptr->type, information.identifiant_oi.type);
      strcpy (ptr->nom, information.identifiant_oi.nom);
      ptr->num_copie = information.identifiant_oi.num_copie;
      ptr->ptr_suivant = NULL;
      if (liste_oi [0] == NULL)
         liste_oi [0] = ptr;
         else anc_ptr->ptr_suivant = ptr;
      anc_ptr = ptr;
      if (fread (information, sizeof (information), 1, fd_information) == 0)
         fin_fichier = TRUE;
```

```
        }

    if (liste_oi [0] == NULL)
       *code_retour = VIDE;

    #if DEBUG_BASE_DE_DONNEES
      printf("-------------------------------------------------------\n");
      printf("Resultats :\n");
      ptr = liste_oi [0];
      while (ptr != NULL)
            {
            printf("Classe    : %s\n", ptr->classe);
            printf("Type      : %s\n", ptr->type);
            printf("Nom       : %s\n", ptr->nom);
            printf("num copie : %d\n\n", ptr->num_copie);
            ptr = ptr->ptr_suivant;
            }
      printf("-------------------------------------------------------\n");
      printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
      printf("-------------------------------------------------------\n");
      printf("<<<TOUS_OBJETS.\n");
    #endif
    }

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

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

premier_oi_pile (nom_pile,identifiant_oi,code_retour,code_erreur)

char              nom_pile [MX_LG_NOM + 1];
st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{
int               max;
st_bd_classement  classement;
short             fin_fichier;
short             trouve;
st_bd_information information;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>PREMIER_OI_PILE.\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * RECHERCHE DU DENIER OI INSERE SUR LA PILE                                 *
 *****************************************************************************/

max         = 0;
fin_fichier = FALSE;
rewind (fd_classement);

if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      if (classement.num_clas > max AND
          strcmp (nom_pile, classement.id_ois.nom) == 0 AND
          strcmp ("PILE", classement.id_ois.classe) == 0)
         {
         strcpy (identifiant_oi->classe, classement.id_oi_classe.classe);
         strcpy (identifiant_oi->type, classement.id_oi_classe.type);
         strcpy (identifiant_oi->nom, classement.id_oi_classe.nom);
         identifiant_oi->num_copie = classement.id_oi_classe.num_copie;
         max = classement.num_clas;
         }
      if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
         fin_fichier = TRUE;
      }

/*****************************************************************************
 * SI ON N'A RIEN TROUVE, IL FAUT VERIFIER QUE LA PILE EXISTE                *
 *****************************************************************************/

if (max == 0)
   {
   rewind (fd_information);
   fin_fichier = FALSE;
```

```
   trouve = FALSE;

   if (fread (information, sizeof (information), 1, fd_information) == 0)
      fin_fichier == TRUE;

   while (fin_fichier == FALSE AND trouve == FALSE)
         {
         if (strcmp (information.identifiant_oi.classe, "PILE") == 0 AND
             strcmp (information.identifiant_oi.nom, nom_pile) == 0)
             {
             *code_erreur = PILE_VIDE;
             trouve = TRUE;
             }
             else    if (fread (information, sizeof (information), 1,
                        fd_information) == 0)
                        fin_fichier == TRUE;
          }
   if (trouve == FALSE)
      *code_erreur = PILE_EXISTE_PAS;
   }

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  printf("Classe    : %s\n", identifiant_oi->classe);
  printf("Type      : %s\n", identifiant_oi->type);
  printf("Nom       : %s\n", identifiant_oi->nom);
  printf("Num copie : %d\n", identifiant_oi->num_copie);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<PREMIER_OI_PILE.\n");
#endif
}

/*****************************************************************************/

lire_type_fichier (identifiant_fichier,classe,type,code_retour,code_erreur)

st_identifiant_oi *identifiant_fichier;
char              classe [MX_LG_CLASSE + 1];
char              type [MX_LG_TYPE + 1];
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{
st_bd_classement classement;
short            trouve;
short            fin_fichier;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>LIRE_TYPE_FICHIER.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Classe du fichier : %s\n", identifiant_fichier->classe);
  printf("Type du fichier   : %s\n", identifiant_fichier->type);
  printf("Nom du fichier    : %s\n", identifiant_fichier->nom);
```

```
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER CLASSEMENT                                             *
 *****************************************************************************/

fin_fichier = FALSE;
trouve      = FALSE;

rewind (fd_classement);

if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE AND trouve == FALSE)
      {
      if (strcmp (classement.id_ois.classe, identifiant_fichier->classe) == 0
         AND strcmp (classement.id_ois.type, identifiant_fichier->type) == 0
         AND strcmp (classement.id_ois.nom, identifiant_fichier->nom) == 0)
         {
         trouve = TRUE;
         strcpy (classe, classement.id_oi_classe.classe);
         strcpy (type, classement.id_oi_classe.type);
         }
         else if (fread (&classement, sizeof (classement),1,
fd_classement)==0)
                  fin_fichier = TRUE;
      }

if (trouve == FALSE)
   *code_retour = FICHIER_VIDE;

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  printf("Classe : %s\n", classe);
  printf("Type   : %s\n", type);
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<LIRE_TYPE_FICHIER.\n");
#endif
}

/*****************************************************************************/

pile_objet_interface (classe,nom,identifiant_ois,code_retour,code_erreur)

char              classe [MX_OI_CLASSE + 1];
char              nom [MX_OI_NOM + 1];
st_identifiant_oi *identifiant_ois;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{
```

```
st_bd_information information;
short             trouve;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>PILE_OBJET_INTERFACE.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Classe : %s\n", classe);
  printf("Nom    : %s\n", nom);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER INFORMATION                                            *
 *****************************************************************************/

rewind (fd_information);
trouve      = FALSE;

fread (&information, sizeof (information), 1, fd_information);

while (trouve == FALSE)
      {
      if (strcmp (information.obj_int_classe, classe) == 0 AND
          strcmp (information.obj_int_nom, nom) == 0)
          {
          strcpy (identifiant_ois->classe, information.identifiant_oi.classe);
          strcpy (identifiant_ois->type, information.identifiant_oi.type);
          strcpy (identifiant_ois->nom, information.identifiant_oi.nom);
          identifiant_ois->num_copie = information.identifiant_oi.num_copie;
          trouve = TRUE;
          }
          else fread (&information, sizeof (information), 1, fd_information);
      }

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  printf("Classe de l'OIS    : %s\n", identifiant_ois->classe);
  printf("Type   de l'OIS    : %s\n", identifiant_ois->type);
  printf("Nom    de l'OIS    : %s\n", identifiant_ois->nom);
  printf("Num copie de l'OIS : %d\n", identifiant_ois->num_copie);
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<PILE_OBJET_INTERFACE.\n");
#endif
}

/*****************************************************************************/

numero_copie (identifiant_oie,numero,code_retour,code_erreur)

st_identifiant_oi *identifiant_oie;
int               *numero;
```

```
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   BOOLEAN           trouve;
   BOOLEAN           fin_fichier;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>NUMERO_COPIE.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

trouve       = FALSE;
fin_fichier  = FALSE;

/*****************************************************************************
 * RECHERCHE DU  PLUS GRAND NUMERO DE COPIE D'UN ORIGINAL                    *
 *****************************************************************************/

rewind (fd_information);

while ((trouve == FALSE) AND (fin_fichier == FALSE))
{  if (fread (&info,sizeof(info),1,fd_information) != 1)
   {  fin_fichier = TRUE;
   }
   else
   {  if ((strcmp(identifiant_oie->classe,info.identifiant_oi.classe) == 0)
      AND (strcmp(identifiant_oie->type  ,info.identifiant_oi.type) == 0)
      AND (strcmp(identifiant_oie->nom   ,info.identifiant_oi.nom) == 0)
      AND (*numero < info.identifiant_oi.num_copie))
      {  trouve = TRUE;
         *code_retour = INFERIEUR;
      }
   }
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<NUMERO_COPIE.\n");
#endif
}

/*****************************************************************************/

classement_oi (identifiant_oi,identifiant_ois,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
st_identifiant_oi *identifiant_ois;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_classement  class;
   st_bd_information info;
```



```
/*****************************************************************************
 * VERIFIER QUE L'OIS EXISTE                                                 *
 *****************************************************************************/
```

```
   BOOLEAN           trouve;
   BOOLEAN           fin_fichier;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CLASSEMENT_OI.\n");
#endif

/*****************************************************************************
 * VERIFIER QUE L'OI EXISTE                                                  *
 *****************************************************************************/

*code_retour = NON_CLASSEMENT;
*code_erreur = OI_EXISTE_PAS;

trouve      = FALSE;
fin_fichier = FALSE;

rewind (fd_information);

while ((trouve == FALSE) AND (fin_fichier == FALSE))
{  if (fread (&info,sizeof(info),1,fd_information) != 1)
   {  fin_fichier = TRUE;
   }
   else
   {  if ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) == 0)
      AND (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) == 0)
      AND (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) == 0)
      AND (identifiant_oi->num_copie ==  info.identifiant_oi.num_copie))
      {  trouve = TRUE;
         *code_erreur = OK;
      }
   }
}

/*****************************************************************************
 * VERIFIER QUE L'OIS EXISTE                                                 *
 *****************************************************************************/

if (*code_erreur == OK)
{
   *code_retour = NON_CLASSEMENT;
   *code_erreur = OIS_EXISTE_PAS;

   trouve      = FALSE;
   fin_fichier = FALSE;

   rewind (fd_information);

   while ((trouve == FALSE) AND (fin_fichier == FALSE))
   {  if (fread (&info,sizeof(info),1,fd_information) != 1)
      {  fin_fichier = TRUE;
      }
      else
      {  if ((strcmp(identifiant_ois->classe,info.identifiant_oi.classe) == 0)
         AND (strcmp(identifiant_ois->type  ,info.identifiant_oi.type) == 0)
         AND (strcmp(identifiant_ois->nom   ,info.identifiant_oi.nom) == 0)
         AND (identifiant_oi->num_copie ==   info.identifiant_oi.num_copie))
```

```
         {  trouve = TRUE;
            *code_erreur = OK;
         }
      }
   }
}

/*****************************************************************************
 * VERIFIER LE CLASSEMENT OU NON DE L'OI DANS L'OIS                           *
 *****************************************************************************/

if (*code_erreur == OK)
{
   trouve      = FALSE;
   fin_fichier = FALSE;

   rewind (fd_classement);

   while ((trouve == FALSE) AND (fin_fichier == FALSE))
   {  if (fread (&class,sizeof(class),1,fd_classement) != 1)
      {  fin_fichier = TRUE;
      }
      else
      {  if ((strcmp(identifiant_oi->classe ,class.id_oi_classe.classe) == 0)
         AND (strcmp(identifiant_oi->type   ,class.id_oi_classe.type) == 0)
         AND (strcmp(identifiant_oi->nom    ,class.id_oi_classe.nom) == 0)
         AND (identifiant_oi->num_copie ==   class.id_oi_classe.num_copie)
         AND (strcmp(identifiant_ois->classe,class.id_ois.classe) == 0)
         AND (strcmp(identifiant_ois->type  ,class.id_ois.type) == 0)
         AND (strcmp(identifiant_ois->nom   ,class.id_ois.nom) == 0)
         AND (identifiant_ois->num_copie ==  class.id_ois.num_copie))
         {  trouve = TRUE;
            *code_retour = CLASSEMENT;
         }
      }
   }
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CLASSEMENT_OI.\n");
#endif
}

/*****************************************************************************/

rechercher_oi (code_recherche,classe,type,liste_oi,code_retour,code_erreur)

int                     *code_recherche;
char                    *classe;
char                    *type;
st_liste_identifiant_oi *liste_oi [];
ty_code_retour          *code_retour;
ty_code_erreur          *code_erreur;
```

```
{
st_bd_information       information;
short                   fin_fichier;
short                   ajoute_oi;
st_liste_identifiant_oi *ptr, *anc_ptr;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>RECHERCHE_OI.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Code de recherche : %d\n", *code_recherche);
  printf("Classe            : %s\n", classe);
  printf("Type              : %s\n", type);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER INFORMATION                                            *
 *****************************************************************************/

fin_fichier  = FALSE;
anc_ptr      = NULL;
liste_oi [0] = NULL;

rewind (fd_information);

if (fread (&information, sizeof (information), 1, fd_information) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
      {
      ajoute_oi = FALSE;
      switch (*code_recherche)
            {
            case 1 : if (strcmp(information.obj_trav_classe, "") != 0)
                        ajoute_oi = TRUE;
                     break;
            case 2 : if (strcmp(information.identifiant_oi.classe, classe)==0
                     AND strcmp(information.obj_trav_classe, "") != 0)
                        ajoute_oi = TRUE;
                     break;
            case 3 : if (strcmp(information.identifiant_oi.classe, classe)==0
                     AND strcmp(information.identifiant_oi.type, type)==0
                     AND strcmp(information.obj_trav_classe, "") != 0)
                        ajoute_oi = TRUE;
                     break;
            }
      if (ajoute_oi == TRUE)
         {
         ptr = (st_liste_identifiant_oi *)
               malloc (sizeof (st_liste_identifiant_oi));
         strcpy (ptr->classe, information.identifiant_oi.classe);
         strcpy (ptr->type, information.identifiant_oi.type);
         strcpy (ptr->nom, information.identifiant_oi.nom);
         ptr->num_copie = information.identifiant_oi.num_copie;
```

```
            ptr->ptr_suivant = NULL;

            if (liste_oi [0] == NULL)
               liste_oi [0] = ptr;
               else anc_ptr->ptr_suivant = ptr;
            anc_ptr = ptr;
            }

         if (fread (&information, sizeof(information), 1, fd_information) == 0)
            fin_fichier = TRUE;
         }

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Resultats :\n");
  ptr = liste_oi [0];
  while (ptr != NULL)
        {
        printf("Classe    : %s\n", ptr->classe);
        printf("Type      : %s\n", ptr->type);
        printf("Nom       : %s\n", ptr->nom);
        printf("Num copie : %d\n\n", ptr->num_copie);
        ptr = ptr->ptr_suivant;
        }
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<RECHERCHE_OI.\n");
#endif
}

/*****************************************************************************/

travail_oi (identifiant_oi,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>TRAVAIL_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * VERIFIER QUE L'OI EST SUR UN OUTIL DE TRAVAIL OU NON                      *
 *****************************************************************************/

rewind (fd_information);

do
{  fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
```

```
        (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
        (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
        (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

if (strcmp(info.obj_trav_classe,"TABLE_DE_TRAVAIL") == 0)
{  *code_retour = TABLE_DE_TRAVAIL;
}

if (strcmp(info.obj_trav_classe,"PHOTOCOPIEUSE") == 0)
{  *code_retour = PHOTOCOPIEUSE;
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<TRAVAIL_OI.\n");
#endif
}

/*****************************************************************************/

creer_oi (valeur_oi,code_retour,code_erreur)

st_valeur_oi   *valeur_oi;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_bd_information info;
   st_bd_information info2;
   BOOLEAN           trouve;
   BOOLEAN           fin_fichier;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CREER_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

strcpy(info.identifiant_oi.classe,valeur_oi->classe);
strcpy(info.identifiant_oi.type  ,valeur_oi->type);
strcpy(info.identifiant_oi.nom   ,valeur_oi->nom);
info.identifiant_oi.num_copie =   valeur_oi->num_copie;
strcpy(info.etat,valeur_oi->etat);
strcpy(info.echeance,valeur_oi->echeance);
strcpy(info.etat_contenu,valeur_oi->etat_contenu);
strcpy(info.ordre_clas,valeur_oi->ordre_clas);
strcpy(info.id_original.classe,"");
strcpy(info.id_original.type  ,"");
strcpy(info.id_original.nom   ,"");
info.id_original.num_copie =   0;
strcpy(info.obj_int_classe ,"");
strcpy(info.obj_int_nom    ,"");
strcpy(info.obj_rang_classe,"");
strcpy(info.obj_rang_nom   ,"");
```

```
strcpy(info.obj_trav_classe,"");
strcpy(info.obj_trav_nom   ,"");

/*****************************************************************************
 * SI C'EST UN ORIGINAL D'OIE, LUI ASSOCIER TOUTES SES COPIE                 *
 *****************************************************************************/

if ((strcmp(valeur_oi->classe,"MESSAGE") == 0) OR
    (strcmp(valeur_oi->classe,"FORMULAIRE") == 0) OR
    (strcmp(valeur_oi->classe,"DOCUMENT") == 0))
{
   if (valeur_oi->num_copie == 0)
   {  fin_fichier = FALSE;

      rewind (fd_information);

      while (fin_fichier == FALSE)
      {  offset = ftell(fd_information);
         if (fread (&info2,sizeof(info2),1,fd_information) != 1)
         {  fin_fichier = TRUE;
         }
         else
         {  if ((strcmp(valeur_oi->classe,info2.identifiant_oi.classe) == 0)
            AND (strcmp(valeur_oi->type  ,info2.identifiant_oi.type) == 0)
            AND (strcmp(valeur_oi->nom   ,info2.identifiant_oi.nom) == 0))
            {  strcpy(info2.id_original.classe,valeur_oi->classe);
               strcpy(info2.id_original.type  ,valeur_oi->type);
               strcpy(info2.id_original.nom   ,valeur_oi->nom);
               info2.id_original.num_copie = 0;
               fseek (fd_information,offset,SEEK_SET);
               fwrite (&info2,sizeof(info2),1,fd_information);
            }
         }
      }
   }

/*****************************************************************************
 * SI C'EST UNE COPIE D'OIE, VERIFIER QUE L'ORIGINAL EXISTE                  *
 *****************************************************************************/

   else
   {  trouve        = FALSE;
      fin_fichier   = FALSE;

      rewind (fd_information);

      while ((trouve == FALSE) AND (fin_fichier == FALSE))
      {  if (fread (&info2,sizeof(info2),1,fd_information) != 1)
         {  fin_fichier = TRUE;
         }
         else
         {  if ((strcmp(valeur_oi->classe,info2.identifiant_oi.classe) == 0)
            AND (strcmp(valeur_oi->type  ,info2.identifiant_oi.type) == 0)
            AND (strcmp(valeur_oi->nom   ,info2.identifiant_oi.nom) == 0)
            AND (info2.identifiant_oi.num_copie == 0))
            {  trouve = TRUE;
            }
```

```
            }
         }
         if (trouve == TRUE)
         {  strcpy(info.id_original.classe,valeur_oi->classe);
            strcpy(info.id_original.type  ,valeur_oi->type);
            strcpy(info.id_original.nom   ,valeur_oi->nom);
            info.id_original.num_copie =   0;
         }
      }
}

/*****************************************************************************
 * ENREGISTRER L'OI DANS LA BASE DE DONNEES                                  *
 *****************************************************************************/

fseek (fd_information,0L,SEEK_END);

fwrite (&info,sizeof(info),1,fd_information);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CREER_OI.\n");
#endif
}

/*****************************************************************************/

modifier_oi (valeur_oi,b_etat,b_echeance,b_contenu,b_ordre,code_retour,
                                                          code_erreur)

st_valeur_oi   *valeur_oi;
int            *b_etat;
int            *b_echeance;
int            *b_contenu;
int            *b_ordre;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;

{  st_bd_information info;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>MODIFIER_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * LECTURE ET MODIFICATION DE L'OI                                           *
 *****************************************************************************/

rewind (fd_information);

do
```

```
{  offset = ftell (fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(valeur_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(valeur_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(valeur_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (valeur_oi->num_copie !=  info.identifiant_oi.num_copie));

if (*b_etat == VRAI)
{  strcpy(info.etat,valeur_oi->etat);
}
if (*b_echeance == VRAI)
{  strcpy(info.echeance,valeur_oi->echeance);
}
if (*b_contenu == VRAI)
{  strcpy(info.etat_contenu,valeur_oi->etat_contenu);
}
if (*b_ordre == VRAI)
{  strcpy(info.ordre_clas,valeur_oi->ordre_clas);
}

fseek (fd_information,offset,SEEK_SET);
fwrite (&info,sizeof(info),1,fd_information);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<MODIFIER_OI.\n");
#endif
}

/*****************************************************************************/

lecture_oi (identifiant_oi,info_oi,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
st_valeur_oi      *info_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   BOOLEAN           trouve;
   BOOLEAN           fin_fichier;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>LECTURE_OI.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Identifiant_oi :\n");
  printf("Classe    : %s.\n",identifiant_oi->classe);
  printf("Type      : %s.\n",identifiant_oi->type);
  printf("Nom       : %s.\n",identifiant_oi->nom);
  printf("Num_copie : %d.\n",identifiant_oi->num_copie);
  printf("-------------------------------------------------------\n");
#endif
```

```
*code_retour = EXISTE_PAS;
*code_erreur = OK;

/*****************************************************************************
 * LECTURE DE L'OI                                                           *
 *****************************************************************************/

trouve        = FALSE;
fin_fichier   = FALSE;

rewind (fd_information);

while ((trouve == FALSE) AND (fin_fichier == FALSE))
{  if (fread (&info,sizeof(info),1,fd_information) != 1)
   {  fin_fichier = TRUE;
   }
   else
   {  if ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) == 0) AND
          (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) == 0) AND
          (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) == 0) AND
          (identifiant_oi->num_copie ==  info.identifiant_oi.num_copie))
      {  trouve = TRUE;
         *code_retour = OK;
      }
   }
}

if (trouve == TRUE)
{  strcpy(info_oi->classe,info.identifiant_oi.classe);
   strcpy(info_oi->type  ,info.identifiant_oi.type);
   strcpy(info_oi->nom   ,info.identifiant_oi.nom);
   info_oi->num_copie =   info.identifiant_oi.num_copie;
   strcpy(info_oi->etat        ,info.etat);
   strcpy(info_oi->echeance    ,info.echeance);
   strcpy(info_oi->ordre_clas  ,info.ordre_clas);
   strcpy(info_oi->etat_contenu,info.etat_contenu);
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<LECTURE_OI.\n");
#endif
}

/*****************************************************************************/

lecture_objet (classe_objet,nom_objet,type_objet,code_retour,
               code_erreur)

char           classe_objet [MX_OBJET_CLASSE + 1];
char           nom_objet [MX_OBJET_NOM + 1];
int            type_objet;
ty_code_retour *code_retour;
ty_code_erreur *code_erreur;
```

```
{  st_bd_outil_trav    outil_trav;
   st_bd_obj_rangement obj_rang;
   st_bd_obj_interface obj_int;
   BOOLEAN             trouve;
   BOOLEAN             fin_fichier;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>LECTURE_OBJET.\n");
#endif

*code_retour = EXISTE_PAS;
*code_erreur = OK;

/*****************************************************************************
 * LECTURE DE L'OBJET OUTIL DE TRAVAIL, DE RANGEMENT OU D'INTERFACE          *
 *****************************************************************************/

trouve        = FALSE;
fin_fichier   = FALSE;

switch (type_objet)
{
case OUTIL_TRAVAIL :

       rewind (fd_outil_trav);

       while ((trouve == FALSE) AND (fin_fichier == FALSE))
       {  if (fread(&outil_trav,sizeof(outil_trav),1,fd_outil_trav) != 1)
          {  fin_fichier = TRUE;
          }
          else
          {  if ((strcmp(classe_objet,outil_trav.classe) == 0) AND
                 (strcmp(nom_objet,outil_trav.nom) == 0))
             {  trouve = TRUE;
                *code_retour = OK;
             }
          }
       }
       break;

case OBJET_RANGEMENT :

       rewind (fd_obj_rang);

       while ((trouve == FALSE) AND (fin_fichier == FALSE))
       {  if (fread (&obj_rang,sizeof(obj_rang),1,fd_obj_rang) != 1)
          {  fin_fichier = TRUE;
          }
          else
          {  if ((strcmp(classe_objet,obj_rang.classe) == 0) AND
                 (strcmp(nom_objet,obj_rang.nom) == 0))
             {  trouve = TRUE;
                *code_retour = OK;
             }
          }
       }
       break;
```

```
case OBJET_INTERFACE :

        rewind (fd_obj_int);

        while ((trouve == FALSE) AND (fin_fichier == FALSE))
        {  if (fread (&obj_int,sizeof(obj_int),1,fd_obj_int) != 1)
           {  fin_fichier = TRUE;
           }
           else
           {  if ((strcmp(classe_objet,obj_int.classe) == 0) AND
                  (strcmp(nom_objet,obj_int.nom) == 0))
              {  trouve = TRUE;
                 *code_retour = OK;
              }
           }
        }
        break;
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<LECTURE_OBJET.\n");
#endif
}

/*****************************************************************************/

outil_travail_oi (identifiant_oi,classe,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
char              classe [MX_OT_CLASSE + 1];
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>OUTIL_TRAVAIL_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * ASSOCIER L'OI A UN OUTIL DE TRAVAIL                                       *
 *****************************************************************************/

rewind (fd_information);

do
{  offset = ftell (fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
```

```
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

strcpy (info.obj_trav_classe,classe);
strcpy (info.obj_trav_nom,"");

fseek (fd_information,offset,SEEK_SET);
fwrite (&info,sizeof(info),1,fd_information);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<OUTIL_TRAVAIL_OI.\n");
#endif
}

/*****************************************************************************/

interface_oi (identifiant_oi,classe,nom,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
char              classe [MX_OT_CLASSE + 1];
char              nom    [MX_OT_NOM + 1];
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>INTERFACE_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * ASSOCIER L'OI A OBJET D'INTERFACE                                         *
 *****************************************************************************/

rewind (fd_information);

do
{  offset = ftell (fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

strcpy (info.obj_int_classe,classe);
strcpy (info.obj_int_nom,nom);
```

```
fseek (fd_information,offset,SEEK_SET);
fwrite (&info,sizeof(info),1,fd_information);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<INTERFACE_OI.\n");
#endif
}

/*****************************************************************************/

classer_oi (identifiant_oi,identifiant_ois,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
st_identifiant_oi *identifiant_ois;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{
st_bd_classement  classement;
int               max;
short             fin_fichier;
short             trouve;
st_bd_information information;
long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>CLASSER_OI.\n");
  printf("-------------------------------------------------------\n");
  printf("Arguments :\n");
  printf("Classe    de l'OIS : %s\n", identifiant_ois->classe);
  printf("Type      de l'OIS : %s\n", identifiant_ois->type);
  printf("Nom       de l'OIS : %s\n", identifiant_ois->nom);
  printf("Num copie de l'OIS : %d\n\n", identifiant_ois->num_copie);
  printf("Classe    de l'OI  : %s\n", identifiant_oi->classe);
  printf("Type      de l'OI  : %s\n", identifiant_oi->type);
  printf("Nom       de l'OI  : %s\n", identifiant_oi->nom);
  printf("Num copie de l'OI  : %d\n", identifiant_oi->num_copie);
  printf("-------------------------------------------------------\n");
#endif

*code_retour = *code_erreur = OK;

/*****************************************************************************
 * LECTURE DU FICHIER CLASSEMENT                                             *
 *****************************************************************************/

fin_fichier = FALSE;
max         = 0;
rewind (fd_classement);

if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE)
```

```
      {
      if (strcmp (classement.id_ois.classe, identifiant_ois->classe) == 0 AND
          strcmp (classement.id_ois.type, identifiant_ois->type) == 0 AND
          strcmp (classement.id_ois.nom, identifiant_ois->nom) == 0)

          if (classement.num_clas > max)
             max = classement.num_clas;

      if (fread (&classement, sizeof (classement), 1, fd_classement) == 0)
         fin_fichier = TRUE;
      }
max += 1;

strcpy (classement.id_ois.classe, identifiant_ois->classe);
strcpy (classement.id_ois.type, identifiant_ois->type);
strcpy (classement.id_ois.nom, identifiant_ois->nom);
classement.id_ois.num_copie = identifiant_ois->num_copie;

strcpy (classement.id_oi_classe.classe, identifiant_oi->classe);
strcpy (classement.id_oi_classe.type, identifiant_oi->type);
strcpy (classement.id_oi_classe.nom, identifiant_oi->nom);
classement.id_oi_classe.num_copie = identifiant_oi->num_copie;
classement.num_clas = max;

fseek (fd_classement,0L,SEEK_END);
fwrite (&classement, sizeof (classement), 1, fd_classement);

/*****************************************************************************
 * MODIFICATION DE L'ATTRIBUT ETAT_CONTENU                                   *
 *****************************************************************************/

fin_fichier = FALSE;
trouve      = FALSE;

rewind (fd_information);

offset = ftell(fd_information);
if (fread (&information, sizeof (information), 1, fd_information) == 0)
   fin_fichier = TRUE;

while (fin_fichier == FALSE AND trouve == FALSE)
      {
      if (strcmp(information.identifiant_oi.classe,identifiant_ois->classe)==0
      AND strcmp (information.identifiant_oi.type,identifiant_ois->type)==0
      AND strcmp (information.identifiant_oi.nom, identifiant_ois->nom)==0)
         {
         trouve = TRUE;
         strcpy (information.etat_contenu, "NON VIDE");
         fseek (fd_information,offset,SEEK_SET);
         fwrite (&information, sizeof (information), 1, fd_information);
         }
         else { offset = ftell(fd_information);
                if (fread (&information, sizeof (information), 1,
                 fd_information) == 0)
                 fin_fichier = TRUE;
              }
      }
```

```
#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<CLASSER_OI.\n");
#endif
}

/*****************************************************************************/

ranger_oi (identifiant_oi,classe,nom,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
char              classe [MX_OR_CLASSE + 1];
char              nom [MX_OR_NOM + 1];
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>RANGER_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * ASSOCIER L'OI A UN OBJET DE RANGEMENT                                     *
 *****************************************************************************/

rewind (fd_information);

do
{  offset = ftell(fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

strcpy (info.obj_rang_classe,classe);
strcpy (info.obj_rang_nom,nom);

fseek (fd_information,offset,SEEK_SET);
fwrite (&info,sizeof(info),1,fd_information);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<RANGER_OI.\n");
#endif
}
```

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

replacer (identifiant_oi,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   st_bd_classement  class;
   BOOLEAN           trouve;
   BOOLEAN           fin_fichier;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>REPLACER.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * ON RECHERCHE L'OI DANS LE FICHIER                                         *
 *****************************************************************************/

trouve      = FALSE;
fin_fichier = FALSE;

rewind (fd_information);

do
{  offset = ftell (fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

/*****************************************************************************
 * ON VERIFIE S'IL EST DANS UN OBJET DE RANGEMENT OU D'INTERFACE              *
 *****************************************************************************/

if ((strcmp(info.obj_rang_classe,"") != 0) OR
    (strcmp(info.obj_int_classe,"") != 0))
{  trouve = TRUE;
}

/*****************************************************************************
 * SINON ON VERIFIE QUE L'OI EST CLASSE                                      *
 *****************************************************************************/

rewind (fd_classement);

while ((trouve == FALSE) AND (fin_fichier == FALSE))
{  if (fread (&class,sizeof(class),1,fd_classement) != 1)
```

```
   {  fin_fichier = TRUE;
   }
   else
   {  if ((strcmp(identifiant_oi->classe ,class.id_oi_classe.classe) == 0)
      AND (strcmp(identifiant_oi->type   ,class.id_oi_classe.type) == 0)
      AND (strcmp(identifiant_oi->nom    ,class.id_oi_classe.nom) == 0)
      AND (identifiant_oi->num_copie ==   class.id_oi_classe.num_copie))
      {  trouve = TRUE;
      }
   }
}

/*****************************************************************************
 * SI OUI, ON REPLACE L'OI EN LE DETACHANT DE SON OUTIL DE TRAVAIL            *
 *****************************************************************************/

if (trouve == TRUE)
{
   strcpy(info.obj_trav_classe,"");
   strcpy(info.obj_trav_nom   ,"");

   fseek (fd_information,offset,SEEK_SET);
   fwrite (&info,sizeof(info),1,fd_information);
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<REPLACER.\n");
#endif
}

/*****************************************************************************/

replacer_oi (identifiant_oi,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>REPLACER_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * ON RECHERCHE L'OI DANS LE FICHIER                                         *
 *****************************************************************************/

rewind (fd_information);
```

```
do
{  offset = ftell (fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

/*****************************************************************************
 * ON REPLACE L'OI EN LE DETACHANT DE SON OUTIL DE TRAVAIL                   *
 *****************************************************************************/

strcpy(info.obj_trav_classe,"");
strcpy(info.obj_trav_nom   ,"");

fseek (fd_information,offset,SEEK_SET);
fwrite (&info,sizeof(info),1,fd_information);

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<REPLACER_OI.\n");
#endif
}

/*****************************************************************************/

replacer_const (identifiant_ois,code_retour,code_erreur)

st_identifiant_oi *identifiant_ois;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_classement  class;
   st_bd_information info;
   BOOLEAN           fin_fichier;
   long              offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>REPLACER_CONST.\n");
#endif

*code_retour = OK;
*code_erreur = OK;

/*****************************************************************************
 * ON RECHERCHE CHACUN DES CONSTITUANTS DE L'OIS                             *
 *****************************************************************************/

fin_fichier = FALSE;

rewind (fd_classement);

while (fin_fichier == FALSE)
{  if (fread (&class,sizeof(class),1,fd_classement) != 1)
```

```
    {  fin_fichier = TRUE;
    }
    else
    {  if ((strcmp(identifiant_ois->classe,class.id_ois.classe) == 0)
       AND (strcmp(identifiant_ois->type  ,class.id_ois.type) == 0)
       AND (strcmp(identifiant_ois->nom   ,class.id_ois.nom) == 0)
       AND (identifiant_ois->num_copie ==  class.id_ois.num_copie))
       {

/*****************************************************************************
 * ET POUR CHACUN DE CEUX-CI, ON LE RECHERCHE DANS LE FICHIER                *
 *****************************************************************************/

          rewind (fd_information);

          do
          {  offset = ftell (fd_information);
             fread (&info,sizeof(info),1,fd_information);
          }
          while
            ((strcmp(class.id_oi_classe.classe,info.identifiant_oi.classe)!=0)
          OR (strcmp(class.id_oi_classe.type  ,info.identifiant_oi.type)!=0)
          OR (strcmp(class.id_oi_classe.nom   ,info.identifiant_oi.nom)!=0)
          OR (class.id_oi_classe.num_copie !=  info.identifiant_oi.num_copie));

/*****************************************************************************
 * ET ON REPLACE LE COMPOSANT EN LE DETACHANT DE SON OUTIL DE TRAVAIL        *
 *****************************************************************************/

          strcpy(info.obj_trav_classe,"");
          strcpy(info.obj_trav_nom   ,"");

          fseek (fd_information,offset,SEEK_SET);
          fwrite (&info,sizeof(info),1,fd_information);
       }
    }
}

#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<REPLACER_CONST.\n");
#endif
}

/*****************************************************************************/

detacher_oi (identifiant_oi,code_retour,code_erreur)

st_identifiant_oi *identifiant_oi;
ty_code_retour    *code_retour;
ty_code_erreur    *code_erreur;

{  st_bd_information info;
   st_bd_classement  class;
   st_identifiant_oi id_ois;
```

```
   BOOLEAN          trouve;
   BOOLEAN          fin_fichier;
   long             offset;

#if DEBUG_BASE_DE_DONNEES
  printf(">>>DETACHER_OI.\n");
#endif

*code_retour = OK;
*code_erreur = OK;


/*****************************************************************************
 * ON RECHERCHE L'OI DANS LE FICHIER                                         *
 *****************************************************************************/

rewind (fd_information);

do
{  offset = ftell (fd_information);
   fread (&info,sizeof(info),1,fd_information);
}
while ((strcmp(identifiant_oi->classe,info.identifiant_oi.classe) != 0) OR
       (strcmp(identifiant_oi->type  ,info.identifiant_oi.type) != 0) OR
       (strcmp(identifiant_oi->nom   ,info.identifiant_oi.nom) != 0) OR
       (identifiant_oi->num_copie !=  info.identifiant_oi.num_copie));

/*****************************************************************************
 * ON LE DETACHE DES OBJETS DE RANGEMENT, INTERFACE ET OUTIL DE TRAVAIL      *
 *****************************************************************************/

strcpy(info.obj_int_classe,"");
strcpy(info.obj_int_nom   ,"");
strcpy(info.obj_rang_classe,"");
strcpy(info.obj_rang_nom   ,"");
strcpy(info.obj_trav_classe,"");
strcpy(info.obj_trav_nom   ,"");

fseek (fd_information,offset,SEEK_SET);
fwrite (&info,sizeof(info),1,fd_information);

trouve      = FALSE;
fin_fichier = FALSE;

/*****************************************************************************
 * ON LE DETACHE DE SON OBJET DE CLASSEMENT EVENTUEL                         *
 *****************************************************************************/

rewind (fd_classement);

while ((trouve == FALSE) AND (fin_fichier == FALSE))
{  offset = ftell (fd_classement);
   if (fread (&class,sizeof(class),1,fd_classement) != 1)
   {  fin_fichier = TRUE;
   }
   else
   {  if ((strcmp(identifiant_oi->classe ,class.id_oi_classe.classe) == 0)
```

```
      AND (strcmp(identifiant_oi->type   ,class.id_oi_classe.type) == 0)
      AND (strcmp(identifiant_oi->nom    ,class.id_oi_classe.nom) == 0)
      AND (identifiant_oi->num_copie ==   class.id_oi_classe.num_copie))
      {  trouve = TRUE;
      }
   }
}

if (trouve == TRUE)
{  strcpy(id_ois.classe,class.id_ois.classe);
   strcpy(id_ois.type  ,class.id_ois.type);
   strcpy(id_ois.nom   ,class.id_ois.nom);
   id_ois.num_copie = class.id_ois.num_copie;

   class.num_clas = 0;
   strcpy(class.id_oi_classe.classe,"");
   strcpy(class.id_oi_classe.type,  "");
   strcpy(class.id_oi_classe.nom,   "");
   class.id_oi_classe.num_copie = 0;
   strcpy(class.id_ois.classe,"");
   strcpy(class.id_ois.type,  "");
   strcpy(class.id_ois.nom,   "");
   class.id_ois.num_copie = 0;

   fseek(fd_classement,offset,SEEK_SET);
   fwrite(&class,sizeof(class),1,fd_classement);

/*****************************************************************************
 * MISE A JOUR DE L'ATTRIBUT "ETAT_CONTENU" DE L'OBJET DANS LEQUEL IL ETAIT*
 *****************************************************************************/

   trouve      = FALSE;
   fin_fichier = FALSE;

   rewind (fd_classement);

   while ((trouve == FALSE) AND (fin_fichier == FALSE))
   {  if (fread (&class,sizeof(class),1,fd_classement) != 1)
      {  fin_fichier = TRUE;
      }
      else
      {  if ((strcmp(id_ois.classe,class.id_ois.classe) == 0)
         AND (strcmp(id_ois.type  ,class.id_ois.type) == 0)
         AND (strcmp(id_ois.nom   ,class.id_ois.nom) == 0)
         AND (id_ois.num_copie ==  class.id_ois.num_copie))
         {  trouve = TRUE;
         }
      }
   }

   if (trouve == FALSE)
   {
      rewind (fd_information);

      do
      {  offset = ftell (fd_information);
         fread (&info,sizeof(info),1,fd_information);
```

```
        }
        while ((strcmp(id_ois.classe,info.identifiant_oi.classe) != 0) OR
               (strcmp(id_ois.type  ,info.identifiant_oi.type) != 0) OR
               (strcmp(id_ois.nom   ,info.identifiant_oi.nom) != 0) OR
               (id_ois.num_copie !=  info.identifiant_oi.num_copie));

        strcpy (info.etat_contenu,"VIDE");

        fseek (fd_information,offset,SEEK_SET);
        fwrite (&info,sizeof(info),1,fd_information);
    }
}


#if DEBUG_BASE_DE_DONNEES
  printf("-------------------------------------------------------\n");
  printf("Code retour : %d Code erreur : %d.\n",*code_retour,*code_erreur);
  printf("-------------------------------------------------------\n");
  printf("<<<DETACHER_OI.\n");
#endif
}

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