MINISTERE DES ENSEIGNEMENTS SECONDAIRE, SUPÉRIEUR ET DE LA RECHERCHE SCIENTIQUE

UNIVERSITE POLYTECHNIQUE DE BOBO-D IOULASSO

ECOLE SUPERIEURE D'INFORMATIQUE
01 BP 1091, Tél. (226) 97 27 64
BOBO-DIOULASSO

MINISTERE DE L'ENSEIGNEMENT DE BASE
.....
PROJET D'APPUI A L'ENSEIGNEMENT DE BASE
01 BP 510 Tél. (226) 31 73 00

THEME :
MISE EN LIGNE DES DONNEES STATISTIQUES DU MEBA

Mémoire de fin d'étude

présenté et soutenu publiquement le 22 décembre 2000

pour l'obtention du

Diplôme d'ingénieur de conception en informatique

par

Alassane OUEDRAOGO

Composition du jury :

Président :	Pr Théodore TAPSOBA
Rapporteur :	M. Denis PLANCHAMP
Examinateurs :	M. Ali KABA, directeur du mémoire M. Thierry LAIREZ, maître de stage

Année universitaire 1999/2000

SOMMAIRE

Remerciements

Je pense qu'aucune œuvre ne se réalise sans une passion et une volonté tenaces, mais non plus sans le regard positif de mes proches chez qui je puise la force nécessaire quand la fatigue se fait sentir.

Je remercie la direction et tout le corps enseignant de l'Ecole Supérieure d'Informatique pour leur travail abattu tout le long de notre formation et plus particulièrement Monsieur Ali KABA responsable du département informatique pour sa disponibilité et les conseils incessants dont il a fait preuve tout le long du déroulement de mon mémoire de fin de cycle.

Mes remerciements vont aussi à l'endroit des responsables du Projet d'Appui à l'Enseignement de Base (PAEB) en particulier Monsieur Thierry LAIREZ.

Je remercie aussi toute l'équipe qui avait en charge la réalisation du projet particulièrement Messieurs Antoine CHABERT et Benoît SALLET pour leur franche collaboration.

Pour finir j'adresse mes profonds remerciements à mes parents et amis pour leur soutien moral et financier.

Toutes mes excuses et tous mes remerciements à ceux que j'ai oubliés.

Préface

Le présent document est organisé comme suit :

- une première partie est consacrée à la compréhension de la problématique relatif au sujet du mémoire, ainsi qu'à la démarche méthodologique utilisée pour résoudre le problème;
- une seconde partie traite des solutions conceptuelles et techniques applicables à la problématique;
- Afin une troisième partie se charge de l'implémentation des solutions retenues.

INTRODUCTION

L'Ecole Supérieure d'Informatique (E.S.I.) dans un souci de former des ingénieurs opérationnels, intègre dans le cursus de formation des ingénieurs de conception en informatique, en fin de cycle, la réalisation d'un mémoire de fin d'étude. La réalisation de ce mémoire a pour objectifs pour les futurs ingénieurs non seulement de consolider les connaissances acquises au cours de la formation à travers la résolution d'un problème informatique dont la solution est moins évidente, mais aussi de découvrir le monde de l'entreprise afin de s'adapter au contexte d'entreprise.
L'étude d'une durée de six mois s'est déroulée au Projet d'Appui à l'Enseignement de Base (PAEB) qui est un projet de la coopération française.

I) PROBLÉMATIQUE

I.1) Présentation générale

Préambule

Dans la plupart des organisations publiques ou privés les décideurs éprouvent le besoin de disposer d'informations suffisamment synthétiques à même de faciliter une prise de décision rapide et efficace.
C'est dans ce souci que la Direction des Etudes et de la Planification (DEP) du Ministère de l'Enseignement de Base (MEBA), appuyée par le Projet d'Appui à l'Enseignement de Base (PAEB), s'est dotée d'un système d'information statistiques sur les données de l'éducation de base pour le suivi de l'évolution de certains indicateurs de l'enseignement de base dont le taux de scolarisation qui est l'un des indicateurs essentiels de développement.
Aussi la publication des informations statistiques de l'enseignement de base doit pouvoir se faire à travers les différents médias de communication (surtout Internet par le biais du web) dans le but :

- de faciliter l'accessibilité à l'information à travers le monde entier;
- de favoriser l'échange d'informations avec les différents partenaires de l'éducation.

Motivations de l'étude

Les motivations qui ont guidé l'étude sont nombreuses et variées, parmi lesquelles nous pouvons retenir :

- Une disponibilité permanente d'informations à la fois synthétiques et précises de l'état du système éducatif du Burkina Faso sur Internet;
- Une publication des indicateurs sur le suivi du système éducatif primaire ;
- Une meilleure disponibilité d'outils de pilotage des systèmes éducatifs
  en dotant les structures centrales et déconcentrées des ministères d'outils fiables de décision :
  - pour maîtriser les coûts des systèmes éducatifs
  - pour accroître le rendement interne et externe des systèmes

En améliorant ou créant les liens fonctionnels entre les différents acteurs de l'éducation pour une meilleure coordination interne du système.

En dynamisant les fonctions études des ministères pour une meilleure évaluation des performances des systèmes éducatifs.

- Une meilleure utilisation des ressources humaines, dans un contexte de limitation des recrutements de fonctionnaires (nouveaux « statuts » des enseignants...) et d'augmentation des effectifs élèves (pression démographique, plans nationaux d'action...).

I.2) Présentation générale du projet

Objectifs généraux du projet

Les objectifs principaux du projet se résument à l'amelioration de la production des indicateurs d'évolution ainsi qu'à la diffusion des données collectées.

Pour atteindre ces objectifs une première étape consistera à la conception d'une structure de données, permettant de stocker toutes les données récoltées, qui favorise l'établissement simple des indicateurs d'évolution.

La seconde étape se résume à la réalisation d'un site Internet pour la diffusion des principales informations de l'annuaire ainsi que d'autres informations relatives à l'enseignement de base.

Enfin la troisième étape consiste à la mise au point d'outil de transfert de données entre bases de données.

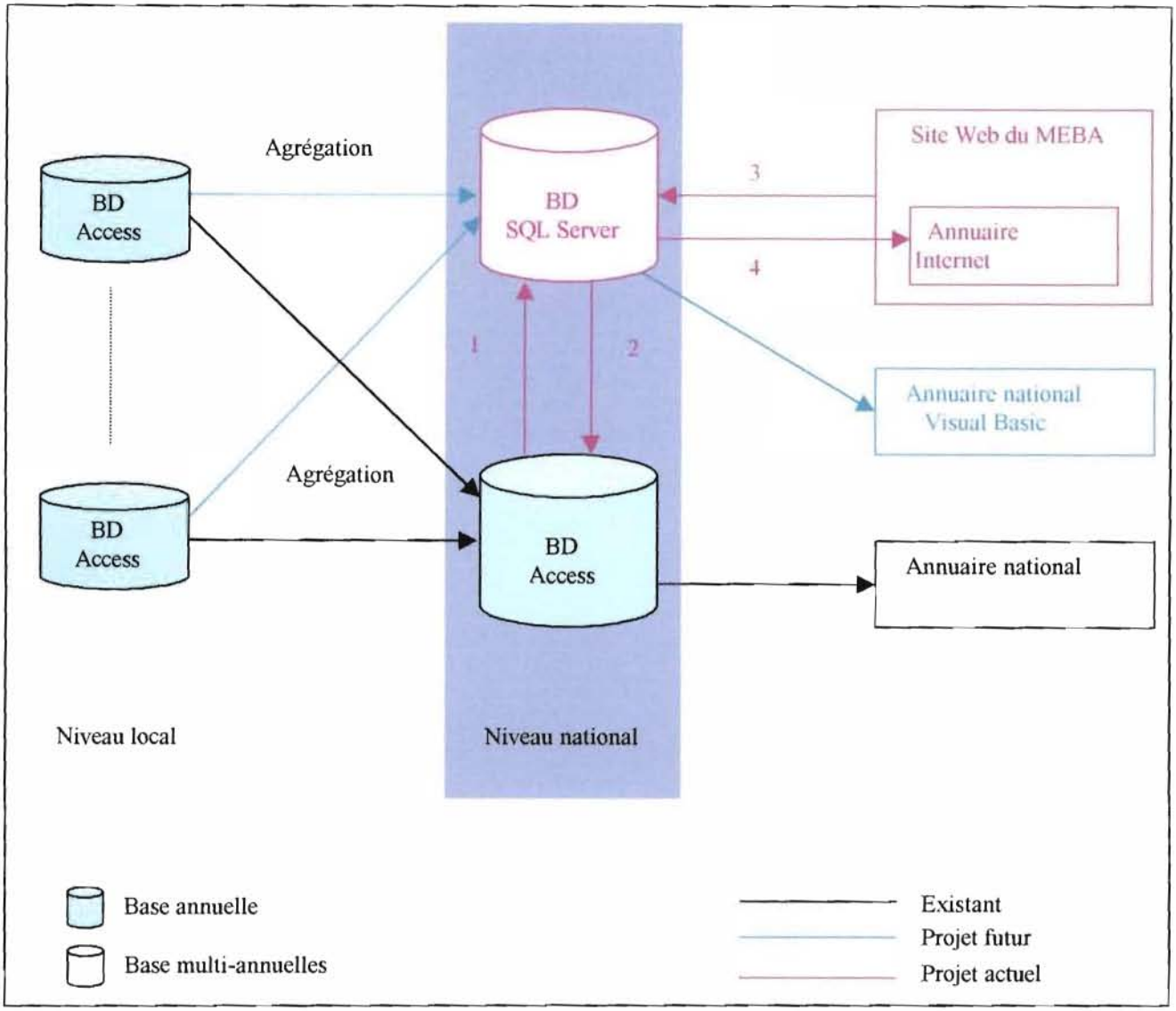


Figure 1 : détail du projet actuel et les extensions à long terme.

Description de l'existant

Le Projet d'Appui à l'Enseignement de Base (PAEB) s'est engagé dans un processus d'informatisation qui a permis le développement d'un certain nombre d'applications basées sur des solutions orientées bureautiques et monopostes.

C'est ainsi qu'une application dénommée "**Application saisie**" permet la saisie des données statistiques collectées au cours d'une année académique. Cette application a été développée en Visual Basic Access (VBA) et utilise une base de données Access.

Il faut noter que la mise au point de l'application a permis de capitaliser quatre années de données sur l'enseignement primaire au Burkina Faso.

Le processus de la collecte des informations est effectué annuellement par les directions regionales et provinciales de l'enseignement de base (DREBA et DEPBA) du ministère de l'enseignement de base (MEBA) auprès des chefs d'établissements. Il s'agit pour ceux-ci de répondre à un questionnaire d'enquête.

La saisie des informations issues des fiches d'enquête est decentralisée au niveau de quelques DREBA grâce à leur informatisation. Les données sont ensuite rassemblées au niveau central

dans une base de données annuelle nationale. La **figure 2** présente la description de l'organisation et du traitement des données.
Chaque zone de saisie dispose d'une copie de la base de données, celles-ci sont, en fin de campagne de saisie, synchronisées au niveau national.

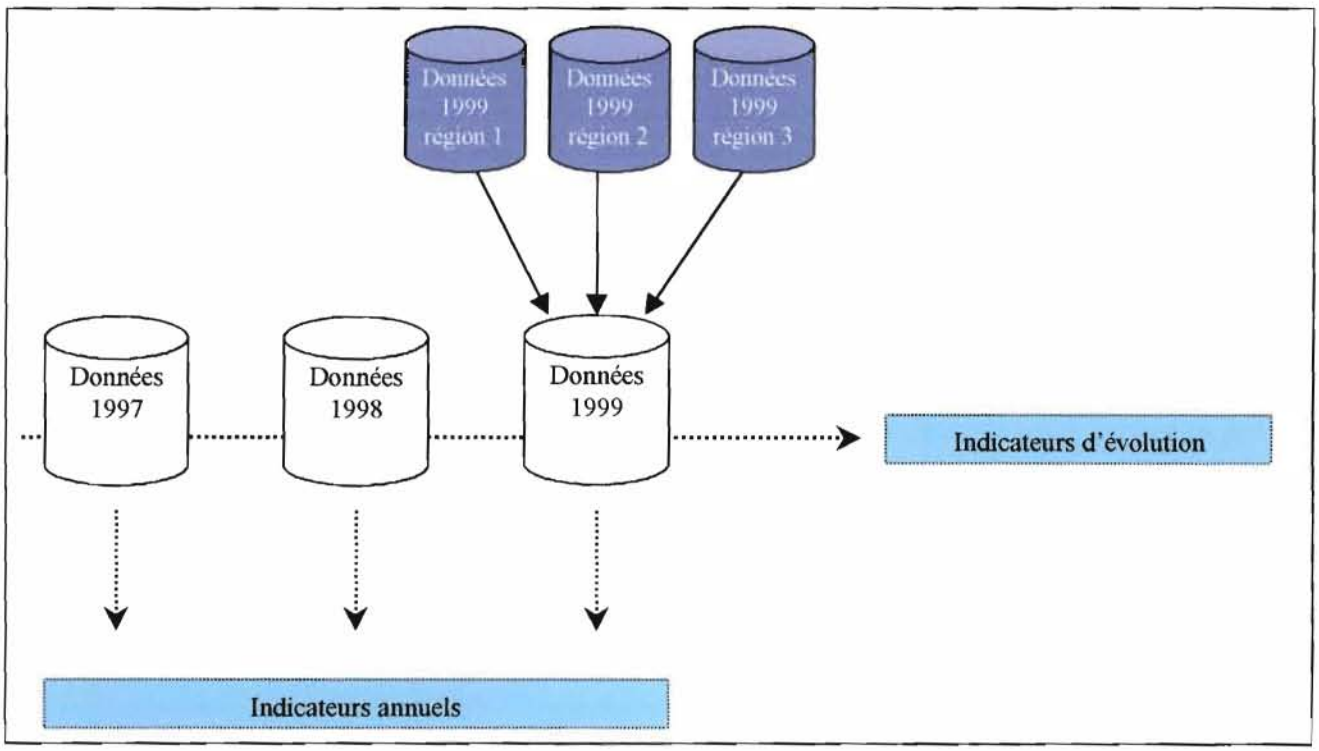


Figure 2 : Organisation des données du système d'information

Une fois que les données ont subi les différents traitements de nettoyage, les indicateurs annuels sont obtenus à l'aide de la même base de données Access puis d'une autre application dénommée "**Annuaire**" qui joue le rôle de frontal pour la production des différents états sur la situation du système éducatif pendant l'année de référence choisie.
Les données recueillies permettent l'élaboration de deux documents : l'annuaire et le tableau de bord.
L'application « Annuaire » permet la production d'un document « Annuaire de l'Education de Base» chaque année rapidement après la saisie des informations. Le Burkina Faso fait partie des pays qui publient leurs indicateurs dans les meilleurs délais. Le lancement de l'enquête a lieu à partir d'octobre puis la saisie se déroule à partir de janvier (jusqu'en juin) et enfin l'annuaire est disponible en septembre. Ceci procure l'avantage de pouvoir adapter la planification de façon annuelle. Ce document présente l'ensemble des tableaux issus des données saisies sur une année et ce depuis trois ans. Il constitue un document de référence mais présente deux inconvénients notables. Premièrement, il manque de données locales puisque les données ne sont disponibles qu'au niveau provincial. Deuxièmement, il est peu adapté à l'analyse du système éducatif du fait de la quantité d'information et de leur présentation.

Depuis l'année 1997/1998, un autre document « le tableau de bord » est créé afin de présenter synthétiquement les informations. Il comprend ainsi des tableaux de l'annuaire de l'année en cours, des graphiques de l'évolution des indicateurs ainsi que des commentaires explicatifs.

Plus lisible que l'Annuaire, c'est la manière dont son contenu est élaboré qui peut être sujet à amélioration.

La **Figure 3** décrit le flux d'informations depuis la saisie jusqu'aux sorties imprimées (tableau de bord et annuaire).

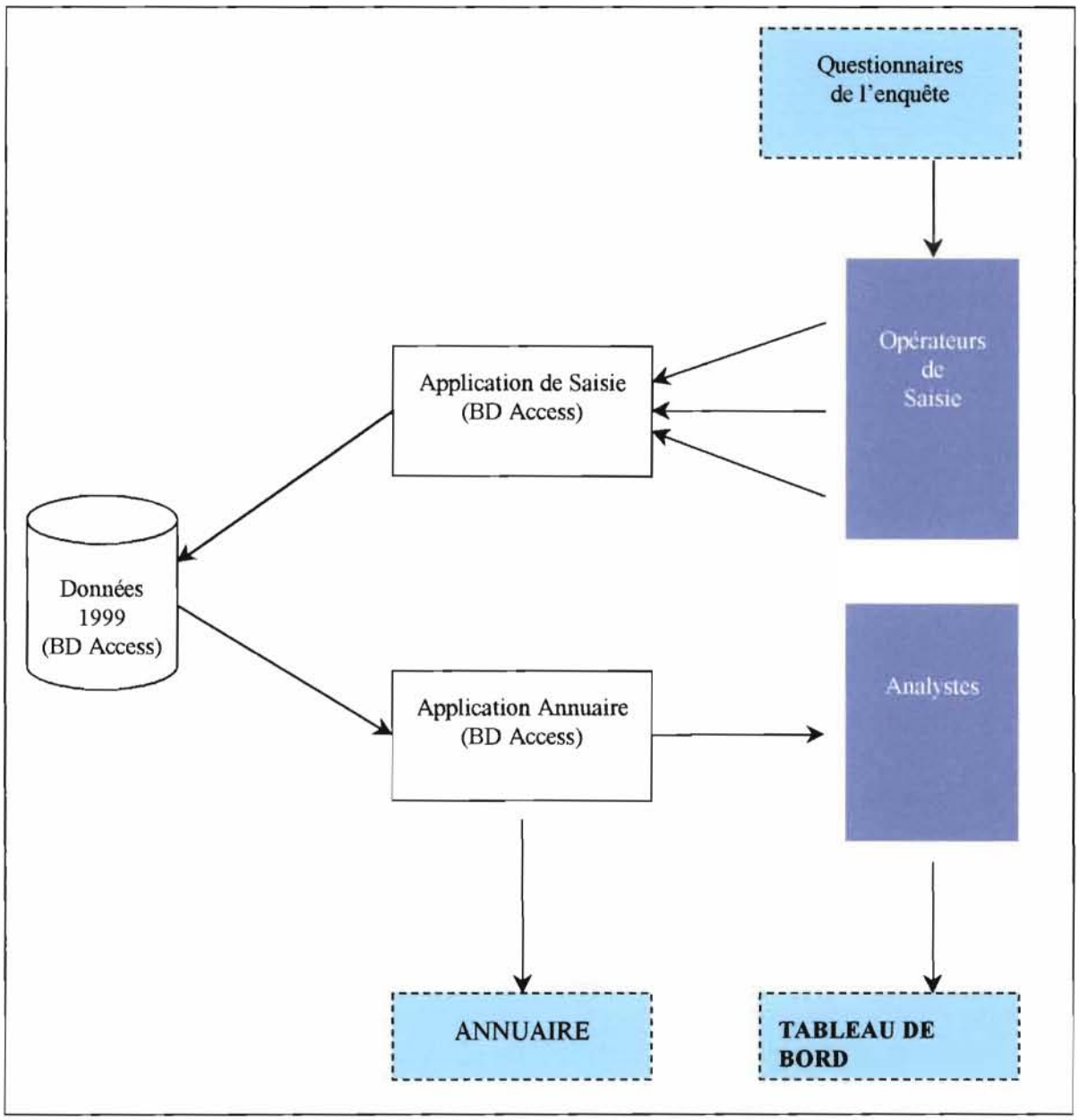


Figure 3 : Flux de données dans le système d'information

I.3) Champ de l'étude

La résolution du problème, qui se résume à la production et à la diffusion d'informations synthétiques resultantes d'une analyse croisée de données statistiques de l'éducation de base, a été confiée à une équipe pluridisciplinaire dont chaque membre avait une composante bien définie du projet à réaliser.

Pour ce qui me concernait j'avais en charge:

- la conception d'une base de données multiannuelle sous SQL Server qui servira de source de données pour alimenter les frontaux du site web ;
- la conception et la réalisation d'un outil de migration de données bidirectionnelle (Microsoft Access vers Microsoft SQL Server) qui favoriserait la mise à jour de la base SQL Server à partir des données sous format Microsoft Access, aussi l'outil doit offrir la possiblité de récupérer les mêmes données de l'environnement Microsoft SQL Server vers un environnement Microsoft Access afin de pourvoir garantir le bon fonctionnement des frontaux développés sous Access pour la réalisation des différents états.

L4) Contraintes à respecter

Matérielles et logicielles

Les contraintes posées par les responsables du projet étaient les suivantes :

- La conservation de l'existant c'est à dire que les différentes applications (l'application de saisie et l'application annuaire) doivent continuer à fonctionner sur le mode actuel, c'est-à-dire via le SGBD Access. Ces solutions informatiques sont pour la plupart des solutions orientées bureautiques du fait de l'absence de compétence en personnel informaticien au sein des DREBA et des DPEBA. Cette absence de ressources qualifiées en informatique ne permet pas d'envisager une saisie décentralisée via Internet;
- Ensuite, la base de données multi-annuelle destinée au site Internet doit être réalisée sur le SGBD SQL Server, choisi pour son rapport coût / performance, afin de bénéficier d'un SGBD assez robuste pour fonctionner avec le site Internet.;
- Tous les développements se feront avec Visual Basic

Équipe de projet

La réalisation de ce projet a été mené par trois personnes, un chef de projet et deux stagiaires.
M. Benoît SALLET étudiant en année de DESS à l'Université d'Angers (France) chargé de réaliser les développements nécessaires à la publication des informations sur le site Internet. La conception du site Web ainsi que la réalisation technique sont les autres domaines d'attribution.
M. Alassane Ouedraogo, en troisième année du cycle des ingénieurs de conception à l'ESI (Ecole Supérieure d'Informatique de Bobo-Dioulasso), a été chargé de la définition de la structure de la base SQL Server et de la réalisation des échanges de données entre la base de données multi-annuelle SQL Server et les bases de données annuelles Access.
M. Ali Kaba chef de département informatique à l'ESI. qui a dirigé ce mémoire.
M. Antoine Chabert, ingénieur informaticien et CSN au PAEB a dirigé le projet en coordonnant les actions des deux stagiaires. De plus, il a pris en charge les modifications nécessaires à la nouvelle représentation géographique et le "nettoyage" des données des différentes bases de données des années précédentes pour permettre un champ d'étude initial de quatre ans.
Enfin, M. Thierry Lairez, planificateur au PAEB et responsable de la composante 3 (cf. **Annexe I**), a assuré la direction scientifique du projet.

II) APPROCHE MÉTHODOLOGIQUE

II.1) Méthodes disponibles

II.1.1 L'approche XML

Le langage de balisage extensible [en anglais Extensible Markup Language] (abrégé XML) décrit une classe d'objets de données appelés documents XML et décrit partiellement le comportement des programmes qui les traitent. XML est un profil d'application ou une forme restreinte de SGML, le langage normalisé de balisage généralisé [norme ISO 8879]. Par construction, les documents XML sont des documents conformes à SGML.

Les documents XML se composent d'unités de stockage appelées entités, qui contiennent des données analysables ou non. Les données analysables se composent de caractères, certaines formant les données textuelles, et le reste formant le balisage. Le balisage décrit les structures logique et de stockage du document. XML fournit un mécanisme pour imposer des contraintes à ces structures.

Un module logiciel appelé **processeur XML** est utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. On suppose qu'un processeur XML effectue son travail pour le compte d'un autre module, appelé **l'application**. Cette spécification décrit le comportement requis d'un processeur XML, c'est à dire la manière dont il doit lire des données XML et les informations qu'il doit fournir à l'application.

Qu'est ce qu'un document XML ?

Un objet de données est un **document XML** s'il est bien formé, tel que précisé dans cette spécification. De plus, un document XML bien formé peut être valide s'il obéit à certaines autres contraintes.

Chaque document XML a une structure logique et une structure physique. Physiquement, le document se compose d'unités appelées entités. Une entité peut appeler d'autres entités pour causer leur inclusion dans le document. Un document commence à la « racine » ou entité document. Logiquement, le document se compose de déclarations, d'éléments, de commentaires, d'appels de caractère et d'instructions de traitement, qui sont indiqués dans le document par du balisage explicite. Les structures logiques et physiques doivent s'imbriquer correctement.

Exemple de document XML avec une déclaration de type de document :

```
<?xml version="1.0"?>
<!DOCTYPE accueil SYSTEM "bonjour.dtd">
<accueil>Bonjour!</accueil>
```

L'identificateur de système « `bonjour.dtd` » donne l'URL d'une DTD pour le document.
Les déclarations peuvent également être données localement, comme dans l'exemple suivant :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE accueil [
<!ELEMENT accueil (#PCDATA)>
]>
<accueil>Bonjour!</accueil>
```

Documents XML bien formés

Un objet textuel est un document XML bien formé si :

1. pris dans son ensemble, il correspond à la production étiquetée `document` ;
2. il obéit à toutes les contraintes de forme données dans cette spécification ;
3. chacune des entités analysables qui est appelée directement ou indirectement dans le document est bien formée.

Document				
[1]	document	::=	prologue élément Divers*	

Les documents XML peuvent, et devraient, commencer par une **déclaration XML** qui indique la version de XML utilisée. L'exemple suivant est un document XML, bien formé mais non valide, parce qu'il ne contient pas de déclaration de définition de type de document.

```
<?xml version="1.0"?>
<accueil>Bonjour!</accueil>
```

Le numéro de version « `1.0` » devrait être employé pour indiquer la conformité à cette version de la spécification ; un document utilisant la valeur « `1.0` » mais ne se conformant pas à cette version de la spécification est en erreur. Le Groupe de travail XML a l'intention de produire des versions ultérieures à la version « `1.0` », mais cette intention ne signifie aucunement un engagement à produire de futures versions de XML ni, si d'autres versions sont produites, à utiliser un plan de numérotation particulier. Puisque de futures versions ne sont pas exclues, cette construction est un moyen de permettre l'identification automatique de la version, si celle-ci devient nécessaire. Les processeurs peuvent signaler une erreur s'ils reçoivent des documents étiquetés avec des versions qu'ils ne connaissent pas.

La fonction du balisage dans un document XML est de décrire sa structure de stockage et sa structure logique et d'associer des paires attributs-valeurs à ces structures logiques. XML fournit un mécanisme, la déclaration de type de document, pour définir des contraintes sur la structure logique et pour gérer l'utilisation des unités de stockage prédéfinies.Un document XML est **valide** si une déclaration de type de document y est associée et si le document est conforme aux contraintes qu'elle exprime.

La déclaration de type de document doit apparaître avant le premier élément dans le document.

Prologue			
[22]	prologue	::=	DéclXML? Divers* (déclTypeDoc Divers*)?
[23]	DéclXML	::=	'<?xml' InfoVersion DéclCodage? DéclDocAuto? S? '?>'
[24]	InfoVersion	::=	S 'version' Égal (' NumVersion ' | " NumVersion ")
[25]	Égal	::=	S? '=' S?
[26]	NumVersion	::=	([a-zA-Z0-9_.:] | '-')+
[27]	Divers	::=	Commentaires | IT | S

Qu'est ce qu'une définition de type de document (DTD) ?

La **déclaration de type de document** XML contient ou désigne des déclarations de balisage qui fournissent une grammaire pour une classe de documents. Cette grammaire est connue comme définition de type de document, ou **DTD**. La déclaration de type de document peut désigner un sous-ensemble externe (un genre spécial d'entités externes) contenant des déclarations de balisage, peut contenir des déclarations de balisage directement dans un sous-ensemble interne ou peut faire les deux. La DTD d'un document se compose des deux sous-ensembles regroupés.

Une **déclaration de balisage** est une déclaration de type d'élément, une déclaration de liste d'attributs, une déclaration d'entités ou une déclaration de notation. Ces déclarations peuvent être contenues entièrement ou partiellement dans des entités paramètres, tel que décrit ci-dessous dans les contraintes de forme et de validité.

Définition de type de document					
[28]	déclTypeDoc	::=	'<!DOCTYPE' S Nom (S IdExterne)? S? ('[' (déclBalisage | AppelEP | S)* ']' S?)? '>'	[	CV : Type de l'élément racine]
[29]	déclBalisage	::=	déclÉlément | DéclListeAtt | DéclEntité | DéclNotation | IT | Commentaires	[	CV : Imbrication stricte des déclarations et des EP]
				[	CF : EP dans le sous-ensemble interne]

Si les sous-ensembles externes et internes sont utilisés, le sous-ensemble interne est considéré comme se produisant avant le sous-ensemble externe. Ceci a pour effet que les déclarations d'entités et de liste d'attributs du sous-ensemble interne ont priorité sur celles du sous-ensemble externe.

Exemples de DTD :

Exemple 1 :

```
<!ELEMENT PERSON ( NAME | REFERENCE | BIRTH | DEATH | BURIAL
                | BAPTISM | NOTE | FATHER | MOTHER | SPOUSE )* >
<!ATTLIST PERSON ID  ID #REQUIRED>

<!--M signifie Masculin, F signifie Féminin -->
<!ATTLIST PERSON SEX (M | F) #IMPLIED>

<!ELEMENT  REFERENCE  EMPTY>
<!ENTITY % sourceref "SOURCE NMTOKEN #REQUIRED">
<!ATTLIST  REFERENCE %sourceref;>

<!ENTITY % event   "(REFERENCE*, PLACE?, DATE?)">
<!ELEMENT  BIRTH   %event;>
<!ELEMENT  BAPTISM %event;>
<!ELEMENT  DEATH   %event;>
<!ELEMENT  BURIAL  %event;>

<!ELEMENT  PLACE   (#PCDATA)>
<!ELEMENT  DATE    (#PCDATA)>

<!ENTITY % personref "PERSON NMTOKEN #REQUIRED">
```

```
<!ELEMENT  SPOUSE  EMPTY>
<!ATTLIST  SPOUSE  %personref;>
<!ELEMENT  FATHER  EMPTY>
<!ATTLIST  FATHER  %personref;>
<!ELEMENT  MOTHER  EMPTY>
<!ATTLIST  MOTHER  %personref;>

<!ELEMENT  NAME    (GIVEN?, SURNAME?)>
<!ELEMENT  GIVEN   (#PCDATA)>
<!ELEMENT  SURNAME (#PCDATA)>

<!ENTITY % xhtml SYSTEM "xhtml/XHTML1-s.dtd">
%xhtml;
<!ELEMENT  NOTE   (REFERENCE*, body)>
```

Exemple 2 :

```
<!ELEMENT PERSON (NAME, EMPLOYEE_ID, PHONE, OFFICE, PHOTO)>
<!ELEMENT NAME          (#PCDATA)>
<!ELEMENT EMPLOYEE_ID   (#PCDATA)>
<!ELEMENT PHONE         (#PCDATA)>
<!ELEMENT OFFICE        (#PCDATA)>
<!ELEMENT PHOTO         EMPTY>
<!NOTATION JPEG SYSTEM "image/jpeg">
<!ATTLIST PHOTO SOURCE ENTITY #REQUIRED>
```

Exemple 3 :

```
<!ENTITY % person SYSTEM "person.dtd">
%person;

<!ELEMENT FAMILY (REFERENCE*, HUSBAND?, WIFE?, CHILD*,
                  MARRIAGE*, DIVORCE*, NOTE*)>

<!ELEMENT  HUSBAND  EMPTY>
<!ATTLIST HUSBAND %personref;>
<!ELEMENT  WIFE     EMPTY>
<!ATTLIST  WIFE     %personref;>
<!ELEMENT  CHILD    EMPTY>
<!ATTLIST  CHILD    %personref;>
<!ELEMENT  DIVORCE  %event;>
<!ELEMENT  MARRIAGE %event;>
```

Déclaration de document autonome

Les déclarations de balisage peuvent affecter le contenu du document, tel qu'il est transmis d'un processeur XML à une application ; les exemples sont des attributs par défaut et des déclarations d'entités. La déclaration de document autonome, qui peut apparaître comme composante de la déclaration de XML, précise s'il y a de telles déclarations externes à l'entité document.

Déclaration de document autonome					
[32]	DéclDocAuto	::=	S 'standalone' Égal (("'" ('yes' | 'no') "'") | ('"' ('yes' | 'no') '"'))	[	CV : déclaration de document autonome]

Dans une déclaration de document autonome, la valeur « yes » indique qu'il n'y a pas de déclaration de balisage externe à l'entité document (dans le sous-ensemble externe de DTD, ou dans une entité paramètre externe appelée dans le sous-ensemble interne) qui affecterait l'information transmise du processeur XML à l'application. La valeur « no » indique qu'il y a ou peut y avoir de telles déclarations de balisage externes. Notez que la déclaration de document autonome précise seulement la présence de *déclarations* externes ; la présence, dans un document, d'appels à des *entités* externes ne change pas son caractère d'autonomie, quand ces entités sont déclarées dans le sous-ensemble interne.

S'il n'y a aucune déclaration de balisage externe, la déclaration de document autonome n'a aucune signification. S'il y a des déclarations de balisage externes mais pas de déclaration de document autonome, la valeur « no » est supposée.
Tout document XML pour lequel `standalone="no"` peut être converti algorithmiquement en document autonome, ce qui peut être souhaitable pour des applications diffusées en réseau.

Contrainte de validité : déclaration de document autonome
La déclaration de document autonome doit avoir la valeur « no » si des déclarations de balisage externes contiennent les déclarations suivantes :

- d'attributs avec des valeurs par défaut, si les élément auxquels s'appliquent ces attributs apparaissent dans le document sans spécification de valeur pour ces attributs, ou
- d'entités (autres que `amp`, `lt`, `gt`, `apos` et `quot`), si des appels à ces entités apparaissent dans le document, ou
- des attributs avec des valeurs sujettes à normalisation, où l'attribut apparaît dans le document avec une valeur qui va changer en raison de la normalisation, ou
- des types d'éléments avec contenu élémentaire, si du blanc apparaît directement dans toute instance de ce type.

Exemple de déclaration XML avec une déclaration de document autonome :

```
<?xml version="1.0" standalone='yes'?>
```

Exemple

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DOCUMENT [

   <!ENTITY ERH "Elliotte Rusty Harold">

   <!ELEMENT DOCUMENT (TITLE, SIGNATURE)>
   <!ELEMENT TITLE (#PCDATA)>
   <!ELEMENT COPYRIGHT (#PCDATA)>
   <!ELEMENT EMAIL (#PCDATA)>
   <!ELEMENT LAST_MODIFIED (#PCDATA)>
   <!ELEMENT SIGNATURE (COPYRIGHT, EMAIL,
LAST_MODIFIED)>
]>
<DOCUMENT>
  <TITLE>&ERH;</TITLE>
  <SIGNATURE>
    <COPYRIGHT>1999 &ERH;</COPYRIGHT>
    <EMAIL>elharo@metalab.unc.edu</EMAIL>
    <LAST_MODIFIED>March 10, 1999</LAST_MODIFIED>
   </SIGNATURE>
</DOCUMENT>
```

Données textuelles et balisage

Le texte se compose de données textuelles et de balisage. Le **balisage** prend la forme de balises ouvrantes, de balises fermantes, de balises d'éléments vides, d'appels d'entité, d'appels de caractère, de commentaires, de délimiteurs de section CDATA, de déclarations de type de document, et d'instructions de traitement.

Tout le texte qui n'est pas du balisage constitue les **données textuelles** du document.

Les caractères esperluète (&) et crochet gauche (<) peuvent apparaître sous leur forme littérale *seulement* quand ils sont utilisés comme délimiteurs de balisage ou dans un commentaire, une instruction de traitement, ou une section CDATA. Ils sont également permis dans la valeur littérale d'une entité dans une déclaration d'entité interne. S'ils sont nécessaires ailleurs, ils doivent être déguisés en utilisant des appels de caractères numériques ou en utilisant les chaînes de caractères « `&` » et « `<` » respectivement. Le crochet droit (>) peut être représenté en utilisant la chaîne de caractères « `>` » et doit, pour compatibilité, être déguisé en utilisant « `>` » ou un appel de caractère quand il apparaît dans la chaîne de caractères « `]]>` » dans du contenu, quand cette chaîne ne marque pas la fin d'une section CDATA.

Dans le contenu des éléments, toute chaîne de caractères ne contenant pas un délimiteur de début de balisage est considéré donnée textuelle. Dans une section CDATA, toute chaîne de caractères ne contenant pas le délimiteur de fin de section CDATA, « `]]>` », est considérée donnée textuelle.

L'apostrophe (') peut être représentée par « `&apos;` », et le caractère guillemet anglais (") par « `&quot;` », afin de permettre à des valeurs d'attribut de contenir ces caractères. (pour plus d'informations cf. Annexe III rubrique grammaire XML en notation BNF).

II.1.2 L'approche UML

Unified Modeling Language (UML) est un langage de modélisation fondé sur les concepts orientés objet. UML est le résultat d'un large conscensus de plusieurs méthodes orientées objet dont les principales sont:

- OMT (Object Modeling Technique) de James Rumbaugh;
- OOD (Object Oriented Design) de Grady Booch;
- OOSE (Object Oriented Software Engineering) de Ivar Jacobson.

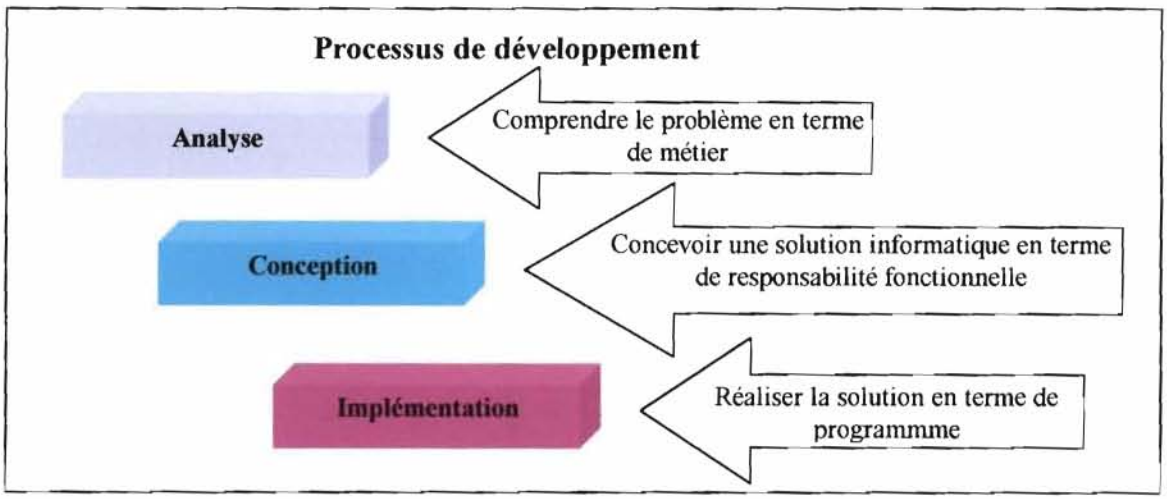
Dans les années quatre vingt, plus de soixante dix méthodes orientées objet ont vu le jour, mais aucune d'elles ne s'est réellement imposée. Cette fragmentation ne milite pas vers l'acceptabilité industrielle de ces technologies orientées objet. C'est ainsi que le besoin d'unification et de normalisation s'imposait et a conduit en 1995 à Unified Method 0.8. Ensuite vint UML 0.9x en juin 1996. En janvier 1997, UML 1.0 fut soumis à l'OMG (Object Management Group) pour normalisation et fut adopté en novembre 1997 dans sa version 1.1.

UML n'est donc pas à l'origine des objets, mais utilise les concepts orientés objet et offre un cadre méthodologique qui unifie les différentes approches en une définition plus formelle pour permettre la modélisation de tous les phénomènes de l'activité de l'entreprise (processus métier, système d'information, systèmes informatiques, composants logiciels, ...) indépendamment des techniques d'implémentation (système automatisé ou non, langage de programmation, ...) mises en œuvre par la suite.

L'approche objet est associée à la modélisation du système d'information en couches qui permet de sérier et hiérarchiser les préoccupations de l'entreprise. De ce fait UML permet le découpage du système d'information de l'entreprise en deux parties :

- le système de métier;
- le système informatique.

UML propose un processus de développement en trois étapes qui sont :
- l'étape de l'analyse;
- l'étape de la conception;
- l'étape de l'implémentation.



Les concepts de base de l'approche objet

Certains concepts de base de l'approche objet sont nécessaires pour mieux appréhender la modélisation avec UML. Parmi ceux-ci nous retiendrons les suivants :

- *Système orienté objet*

L'approche orientée objet est fondée sur le principe suivant: tout système orienté objet est en fait une société d'objets qui coopérent. Pour coopérer, les objets utilisent des messages qu'ils s'envoient entre eux par divers mécanismes dépendant de l'environnement de mise en œuvre.
Tout système au sens large avec un contour bien défini peut faire l'objet d'une modélisation orientée objet.

- *Objet*

Un objet est un membre d'un système orienté objet et est défini dans son domaine par:
- Une identité qui constitue le moyen d'identifier l'objet par rapport aux autres objets du système. Chaque objet dans un système doit avoir une identité;
- Un comportement qui définit la manière dont l'objet réagit aux autres messages qui lui parviennent de son environnement;
- Un état qui définit l'une des possiblités dans laquelle un objet peut se trouver en un instant donné de sa vie.

- *Classe*

Une classe est une description abstraite d'un ensemble d'objets ayant des propriétés similaires, un comportement commun, des relations communes avec d'autres objets et des sémantiques communes.

- Les critères suivants impliquent une agrégation:
  - une classe fait partie d'une autre classe;
  - les valeurs d'attributs d'une classe se propagent dans les valeurs d'attributs d'une autre classe;
  - une action sur une classe implique une action sur une autre classe;
  - les objets d'une classe sont subordonnés aux objets d'une autre classe;
- l'agrégation peut être multiple, comme l'association.

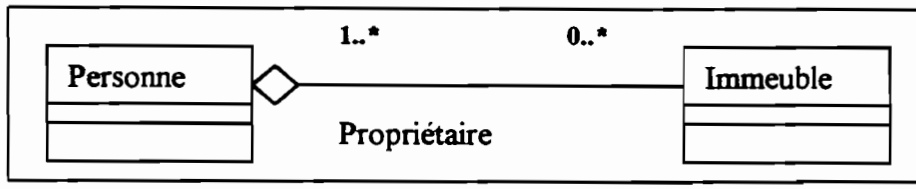


Figure 5 : représentation d'une agrégation

- *Notion de composition*

- La composition porte sur les attributs d'une classe.
- La notation par composition s'emploie dans un diagramme de classes lorsqu'un attribut participe à d'autres relations dans le modèle .

L'exemple suivant montre que la classe Voiture peut être décomposée en des classes d'éléments constituants l'objet voiture. C'est ainsi qu'une voiture est composée d'un moteur qui à son tour est composé d'un cylindre, d'un carburateur,...

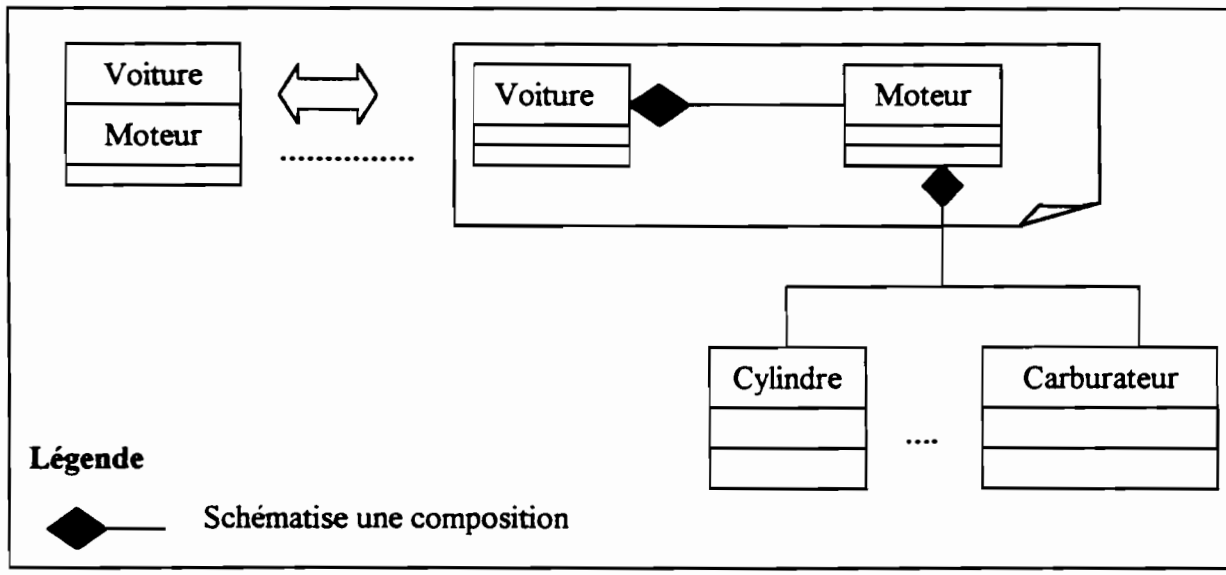


Figure 6 : représentation d'une composition

- la composition implique une contrainte sur la valeur de la multiplicité du côté de l'agrégat : elle ne peut prendre que les valeurs 0 ou 1.
- La valeur 0 du côté du composant correspond à un attribut non renseigné.

- *Notion d'héritage*

Le mécanisme d'héritage permet à une définition de classe d'hériter de tout ou partie des définitions d'une autre classe. Il est donc en général associé une hiérarchie de types dont la sémantique peut varier. On distingue ainsi trois grandes familles de hiérarchies liées à l'héritage:

- l'héritage par substitution;
- l'héritage par inclusion;
- l'héritage par spécialisation.

- *Notion de généralisation*

- La généralisation est une forme d'abstraction par laquelle un ensemble d'objets appelés spécialisés est vu comme un objet unique dit générique.
- Un lien d'héritage entre deux classes établit une relation de spécialisation / généralisation entre chaque objet de la première classe (objet spécialisé) et un objet de la deuxième classe (objet généralisé).
- La généralisation s'applique principalement aux classes, aux paquetages et aux cas d'utilisations.
- Dans le cas des classes, la relation de généralisation signifie *"est un ou est sorte de"* .

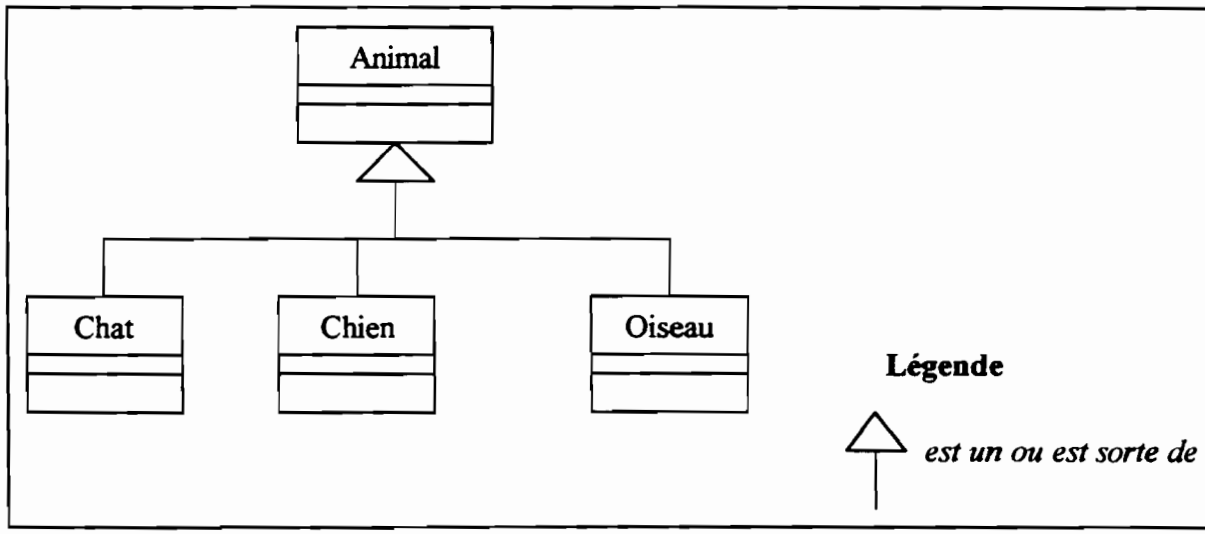


Figure n° 7 : représentation d'une relation d'héritage

Concepts de base d'UML

UML est un langage de modélisation constitué par un ensemble d'éléments de modélisatioin et de diagrammes.

- Les éléments de modélisation représentent toutes les propriétés du langage. Ils ne sont pas limités par le symbolisme graphique utilisé. Ils doivent être clairs et précis.
- Les diagrammes constituent l'autre composant d'UML. Ils en constituent l'expression visuelle et graphique. Un diagramme n'est pas un modèle, mais une projection sur le modèle, une perspective sur le modèle. UML définit neuf diagrammes, chacun permettant de présenter une perspective de l'architecture du système.

- *le diagramme de classes*

Le diagramme de classes est un graphe d'éléments de modélisation. Un diagramme de classes peut contenir aussi des types, des paquetages, des relations et même des instances, tels que des objets et des liens. Il permet la représentation de la structure statique des classes et de leurs relations.

- *le diagramme de séquence*

Le diagramme de séquence montre les interactions, entre les objets, arrangées en séquence dans le temps. En particulier, il montre les objets participant dans l'interaction par leurs "lignes de vie" et les messages qu'ils échangent ordonnancés dans le temps.
Les diagrammes de séquence sont utiles pour comprendre, documenter et expliquer des modèles et non pour les implémenter. Ils doivent être clairs pour faciliter la communication.

- *le diagramme d'activités*

Le diagramme d'activités représente l'état de l'exécution d'un mécanisme, sous la forme d'un déroulement d'étapes regroupées séquentiellement dans des branches parallèles de flot de contrôle. Le diagramme d'activités est organisé par rapport aux actions et destiné à représenter le comportement interne d'une méthode ou d'un cas d'utilisation.

- *le diagramme de collaboration*

Le diagramme de collaboration montre les interactions entre les objets et leurs liens. Contrairement au diagramme de séquence, un diagramme de collaboration montre les relations entre les objets. En revanche, un diagramme de collaboration n'intègre pas la dimension du temps.

- *le diagramme d'objets*

Le diagramme d'objets permet la représentation des objets et de leurs relations. Il facilite la compréhension des structures de données complexes. Un objet composite est un objet composé d'autres objets et représentant des instances de classes composites.

- *le diagramme des cas d'utilisation*

Le diagramme des cas d'utilisation permet de définir le comportement d'un système ou la sémantique de toute autre entité sans révéler la structure interne de l'entité. Chaque cas d'utilisation spécifie une séquence d'actions, y compris des variantes, que l'entité réalise, en interagissant avec les acteurs de l'entité.
La responsabilité d'un cas d'utilisation est de spécifier un ensemble d'instances du cas d'utilisation où une instance de cas d'utilisation représente une séquence d'actions que le système réalise et qui fournit un résultat observable par un acteur particulier.

- *le diagramme des transitions d'états*

Le diagramme des transitions d'états représente le comportement d'une classe en terme d'états. A chaque classe est associée un automate qui suit le comportement de la classe.

- *le diagramme des composants*

Le diagramme de composants décrit les éléments physiques, appelés modules, et leurs relations dans l'environnement de réalisation. Il montre les choix de réalisation et représente toutes les sortes d'éléments physiques qui entrent dans la fabrication des applications informatiques.
On distingue trois types de composants :

Niveau	Préoccupation	Modèles	
		Données	Traitements
Conceptuel	Que doit-on faire ?	**MCD** : Modèle Conceptuel de Données. Formalise la mémoire de l'entreprise et les règles de gestion que cette structure doit respecter.	**MCT** : Modèle Conceptuel des Traitements Formalise les règles de gestion qui dictent la réaction de l'entreprise face à des sollicitations externes.
Logique ou Organisationnel	Qui fait quoi? où ? quand ?	**MLD** : Modèle Logique de Données Définit la réalisation logique de la mémoire de l'entreprise qui tient compte de l'état de l'art en matière technique de gestion des données.	**MOT** : Modèle Organisationnel des Traitements Complète le MCT en prenant en compte les choix d'organisation répond aux questions qui? où? quand?
Physique ou Opérationnel	Comment ?	**MPD** : Modèle Physique de Données Définit les structures de données qui seront utilisées en tenant compte des systèmes logiciels et matériels existants.	**MOpT** : Modèle Opérationnel des Traitements Propose la structure du logiciel (c.à.d les unités de traitement).

Langage de communication

Il offre un formalisme adéquat pour décrire les aspects pertinents du système. Il permet en quelque sorte de décrire le perçu. Cette description doit permettre de dialoguer entre les différents intervenants.

Présentation du formalisme du MCD

Le formalisme du MCD possède une représentation graphique comportant trois concepts types de base : les concepts de *propriété type, entité type* et de *relation type.*

Notion de propriété type et de dictionnaire de données

La notion de propriété ou propriété type permet de conceptualiser dans le système d'information, un type d'information élémentaire du domaine d'étude.

Exemples de propriétés type

nom d'un établissement,

date d'ouverture de l'établissement.

- la spécification qui correspond aux interfaces de classes;
- le corps qui est la réalisation de la classe;
- la spécification générique qui est une classe paramétrable.

Par défaut, chaque classe est realisée par deux composants : la spécification et le corps.

- *le diagramme de déploiement*

Le diagramme de déploiement montre la disposition physique des différents matériels (les noeuds) qui entrent dans la composition d'un système et la répartition des programmes exécutables sur ces matériels.

II.1.3 La méthode Merise

Démarche méthodologique

Nous distinguons trois cycles dans la démarche méthodologique :

Le cycle de vie

Il structure le procédé de développement du logiciel en étapes qui s'enchaînent. Les principales étapes du cycle de vie sont les suivantes :

- L'analyse des besoins ;
- La conception détaillée ;
- La conception technique.

Le cycle de décision

Il rend compte des choix qui doivent être faits durant le cycle de vie. Ces choix s'articulent autour d'une hiérarchie de décisions dont les plus importantes sont :

La définition des orientations majeures concernant le système de gestion, l'organisation et les solutions technologiques ;

Le découpage du système en sous-systèmes ou domaines. En effet, lorsque le domaine d'étude est très complexe, on peut le décomposer en domaines ayant peu de communication entre eux ou ayant des finalités complémentaires.

Le cycle d'abstraction

Il consiste à hiérarchiser les informations du système selon leur niveau de stabilité. Ce sont :

- Le niveau conceptuel : c'est le niveau le plus stable du système d'information. Il décrit ce que doit faire le système indépendamment de toute contrainte d'organisation ou technique ;
- Le niveau logique ou organisationnel : décrit qui fait quoi et où dans le système d'information ;
- Le niveau physique ou opérationnel : c'est le résultat des décisions techniques qui ont été prises en fonction des décisions et des contraintes techniques (performances, capacités des mémoires secondaires, temps de réponse, etc.).

Modèles

Ils offrent les concepts nécessaires pour guider et formaliser le raisonnement lors de l'identification des aspects pertinents du système. Les principaux modèles qui sont utilisés dans ce document sont :

Une propriété est caractérisée par :

- Son nom (exemple : nom, code, ...);
- Son type (liste des valeurs qu'elle peut prendre);
- Son format de présentation (qui détermine par exemple sa longueur).

Le dictionnaire de données définit l'ensemble des propriétés. Il ne doit contenir ni synonymes, ni polysèmes.

Notion d'entité type

L'entité type permet de modéliser un ensemble d'objets de même nature, concrets ou abstraits, qui ont un intérêt pour le domaine considéré. L'entité type exprime un type, une classe, un ensemble dont les éléments sont appelés *occurrences individu type*. L'entité type est symbolisée graphiquement comme suit :

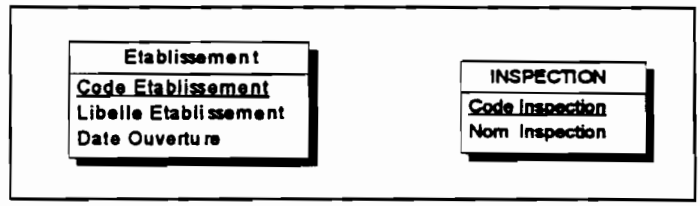


Figure 8 : Entité type

On appelle occurrence d'entité, un objet particulier de la classe d'objet définie par l'entité considérée. Chaque entité doit avoir une propriété particulière appelée identifiant dont les valeurs permettent d'identifier les occurrences de l'entité.

Exemples :

Pour les entités du modèle précédent, les identifiants sont les suivants :

Etablissement : Code Etablissement

Inspection : Code Inspection

Notion de relation type (association)

La relation type modélise un ensemble d'associations de même nature entre les occurrences d'entité (de types différents ou du même type) qui ont un intérêt pour le domaine d'étude. La relation type est représentée graphiquement comme suit :

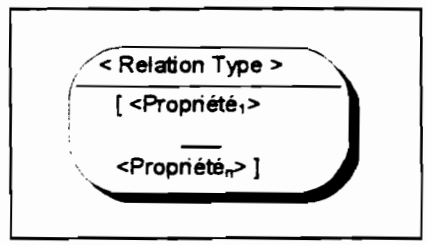


Figure 9 : La relation type.

Une relation n'a pas d'existence propre. La relation type n'est définie qu'à travers les entités qui la composent.

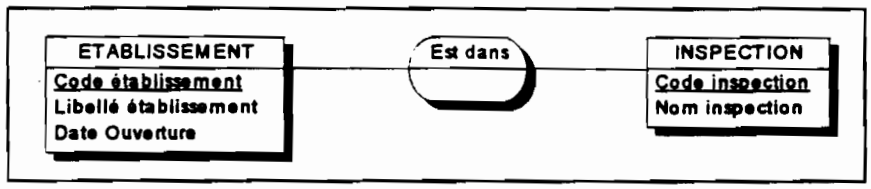


Figure 10 : Entités types reliés par une relation type.

Notion de cardinalité

Le terme cardinalité, traduit la participation des occurrences d'une entité type aux occurrences d'une relation type. Cette participation s'exprime en associant à chaque "patte" liant une entité à une relation deux valeurs : la *cardinalité minimum* et la *cardinalité maximum*.

La cardinalité minimum correspond au nombre minimum de fois qu'une occurrence d'une entité type participe aux occurrences de la relation type.

La cardinalité maximum indique le nombre maximum de fois qu'une occurrence de l'entité type participe aux occurrences de la relation type

Exemple :

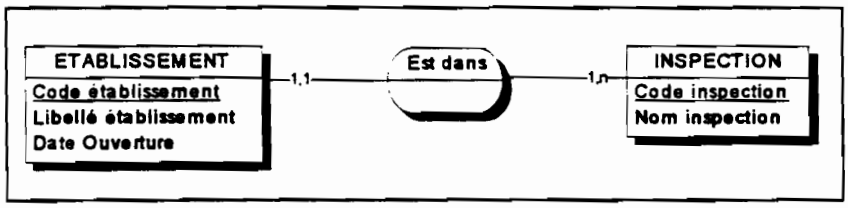


Figure 11 : Cardinalités de la relation <Est dans>.

Présentation du formalisme du MLD

Le modèle logique est une représentation du modèle conceptuel en fonction des techniques du moment, autrement dit de l'état de l'art technologique des matériels et logiciels présents sur le marché.

Actuellement, en matière de logiciel de bases de données, nous assistons au règne des logiciels dits relationnels. Ce sont les modèles de données dit relationnels que nous allons traiter dans les sections ci-dessous.

Concepts de base

Le modèle relationnel s'appuie sur trois concepts de base : le *domaine*, la *relation* (appelée couramment *table)* et *l'attribut.*

Le domaine est l'ensemble des valeurs que peut prendre une information ou donnée.

La table (ou relation), concept central du modèle, peut être définie grossièrement comme un tableau de données. Chaque colonne de ce tableau sera un attribut de la relation qui pourra

prendre les valeurs de son domaine. Les lignes de ce tableau, occurrences de la relation, sont appelées des *tuples* ou des n-*uplets*. Dans les schémas, on utilisera pour elle la même représentation graphique que celle correspondant aux entités du modèle conceptuel.

# code	libellé	Prix unit	quantité

Figure n°12 : représentation graphique d'une table

Tout comme dans le modèle conceptuel des identifiants ont été utilisés pour différencier des occurrences d'individu, dans le modèle relationnel des tuples pourront être aussi identifiés de façon unique en utilisant une *clé primaire simple* (un seul attribut) *ou clé primaire composée* (plusieurs attributs). On soulignera cette clé primaire dans le schéma de la table.

Pour relier deux tables, on utilisera une contrainte référentielle qui ne sera autre chose qu'un lien sémantique défini entre les deux tables A et B et réalisé par la duplication d'une clé primaire d'une table dans l'autre. Cette clé dupliquée est appelée *clé externe* (ou étrangère ou voisine).

Présentation du formalisme du MPD

Le Modèle Physique de Données va permettre d'implanter en machine, l'ensemble des données issues du niveau logique en tenant compte des ressources physiques (gestionnaire de données, matériel, support, etc).

Dans les environnements de données relationnels, les données sont décrites avec un langage de description des données et utilisées via un langage de manipulation des données. De plus en plus, de logiciels permettent de s'abstraire des descriptifs d'organisation des objets informatiques (fichiers, index, listes, ...), mais aussi des langages explicites de description de données. C'est le cas des SGBD actuels (comme SQL Server, Oracle, Access, ...) qui offrent de puissants outils visuels de description et d'accès aux données.

II.2) Démarche utilisée

Méthode

Pour mener à bien la résolution du problème qui nous a été soumis nous serons amené à combiner plusieurs méthodes:

1- c'est ainsi que pour la conception et l'implantation de la base de données SQL Server 7 devant servir de source de données au site nous utiliserons la méthode Merise pour la modélisation des données.
La méthode Merise est une approche systémique d'analyse et de conception de systèmes d'informations. Le choix de la méthode Merise se justifie par le fait qu'elle est une approche orientéee données et traitements. Aussi les formalismes permettent la description des données de manière indépendante des traitements (cf. le " modèle individuel" de H. Tardieu et le "modèle Entité-Relation" de P. Chen qui, tous les deux, appartiennent à la famille des formalismes "Entités/Relations" et le "modèle relationnel", dû à Codd).

2- Pour la description de l'architecture de certains composants ou modules de l'application nous opterons pour l'approche UML à travers le formalisme des diagrammes des classes ou des composants qui facilite une meilleure lisibilité et une compréhension rapide.
UML possède des concepts qui permettent l'expression de façon claire et précise du processus de développement des applications.
L'implémentation est basée sur l'approche XML en ce qui concerne la structuration des différents fichiers dérivant les différentes bases de données (source et destination) qui serviront de sources de données textuelles pour alimenter la base de données destination.
L'approche XML offre une possibilité de pouvoir manipuler dans une seule entité les structures de données et les données et cela réduit considérablement le nombre d'objets à gérer par l'application en cours de réalisation.

3- Une description algorithmique sera utilisée pour traduire l'implémentation des différents traitements.

Concepts

Les concepts utilisés sont respectifs aux méthodes ou approches suivantes:

- les concepts de la méthode Merise
  - entité / association (cf. la présentation de Merise)
  - modèle logique de données (cf. la présentation de Merise)
  - modèle physique de données (cf. la présentation de Merise)
- les concepts de l'approche XML
- notion de DTD (cf. la présentation de XML)
- notion de Document XML (cf. la présentation de XML)
- notion de documents valides (cf. la présentation de XML)
  - notion de documents bien formés (cf. la présentation de XML)
- les concepts de la méthode UML
  - les diagrammes des classes

Un diagramme de classes montre uniquement des aspects statiques du modèle et fait abstraction des aspects dynamiques ou temporels, même si les éléments du diagramme de classes peuvent avoir un comportement dynamique important.

♦ *Les classes*

Comme enoncé précédemment, une classe est représentée par un rectangle en trois compartiments : la partie supérieure contient le nom de la classe et éventuellement un stéréotype, la partie mediane contient les attributs et la partie inférieure contient les opérations.

La syntaxe de déclaration d'un attribut est la suivante :
nom_attribut : type_attribut=valeur_initiale.
On peut aussi définir des attributs dérivés qui sont calculés à partir des éléments existants (exemple du périmètre ou de la surface pour une classe quadrilatère).
La syntaxe est : / nom_attribut_dérivé.

La syntaxe de déclaration d'une opération est la suivante :
nom_opération (nom_argument : type_arg=valeur_défaut,...) : type_retourné.

Il existe trois (3) niveaux de visibilité définissant la portée des attributs et des opérations:
+ public : l'élément est visible par toutes les autres classes;
- privé : l'élément n'est visible que dans la classe où il se trouve;
protégé : l'élément est visible par la classe et ses sous-classes;

On peut aussi définir des variables et des opérations de classes : leur nom est souligné dans la représentation.

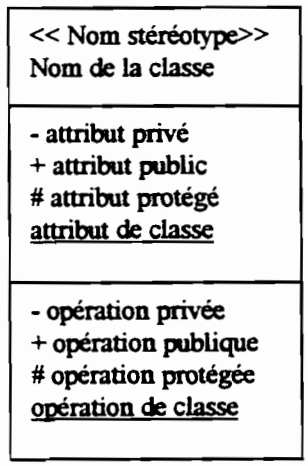


Figure 13 : représentation d'une classe

♦ ***Interface de classes***

Une interface utilise un type pour décrire le comportement visible d'une classe, d'un composant, ou d'un paquetage. L'interface de classe est représentée à l'aide d'un cercle qui lui est associé ou par une classe avec le stéréotype <<interface>>.

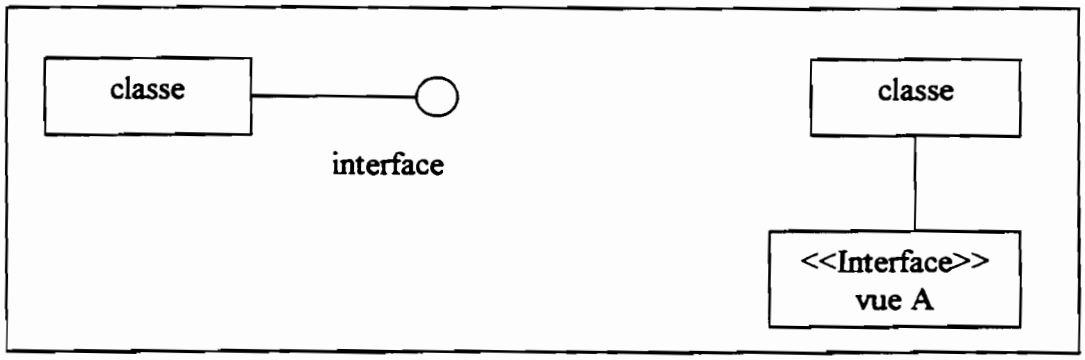


Figure 14 : représentations d'une interface

- ***Les classes paramétrables***

Il s'agit de modèles de classes qui permettent d'obtenir des classes réelles par instanciation.Les classes paramétrables sont surtout utilisées en conception détaillée, pour incorporer par exemple des composants réutilisables. Une classe paramétrable est représentée de la façon suivante :

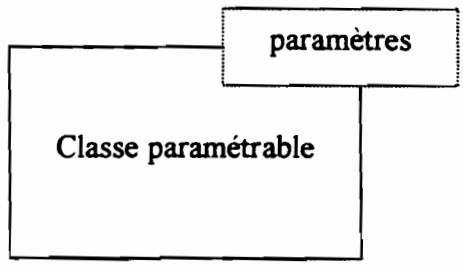


Figure 15 : représentation d'une classe paramétrable

- ***Les associations***

Une association représente une relation structurelle entre des classes d'objets. Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées.

On précise également la cardinalité de chaque classe qui indique le nombre d'occurrence d'objets de la classe qui participent à la relation. Ces cardinalités sont définies comme suit:

- 1 : un et un seul;
- 0..1 : zéro ou un;
- M..N : de M à N;
- * : plusieurs;
- 0..* : de zéro à plusieurs;
- 1..* : de un à plusieurs.

Exemple : association de filiation

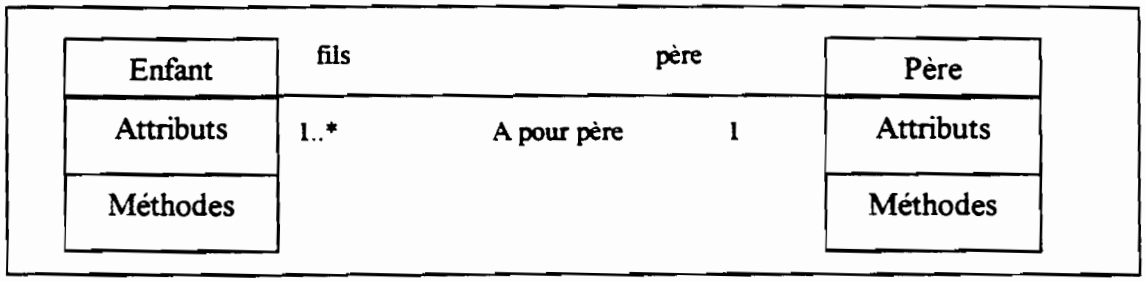


Figure 16 : association entre classes

- ***L'agrégation***

  L'agrégation est le type de relation "partie-de" définie entre les instances de classes. Il existe deux types d'agrégation :

  - la composition stricte : une instance composante appartient à une et une seule instance composite à un instant donné. Cette appartenance peut varier dans le temps et la destruction du composite entraîne la destruction du composant. Elle est représentée par un losange noir.
  - L'agrégation partagée : une instance composante peut appartenir à plusieurs instances composites à la fois. Elle est représentée par un losange en blanc du côté de l'agrégat.

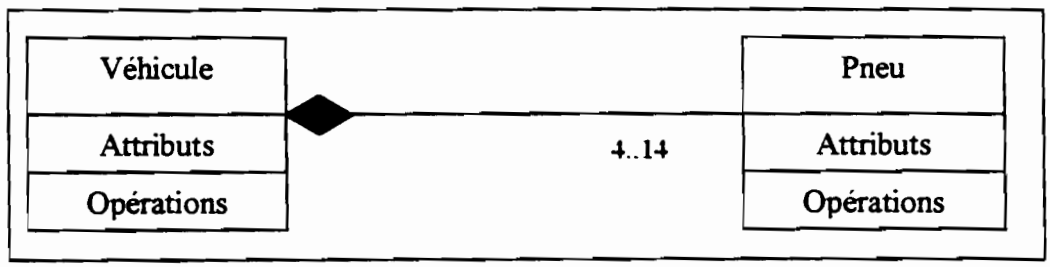


Figure 17 : représentation d'une agrégation

- ***La généralisation***

Elle est représentée par une flèche (triangle vide) qui pointe de la classe la plus spécialisée vers la classe la plus générale et exprime un lien d'héritage entre les classes.

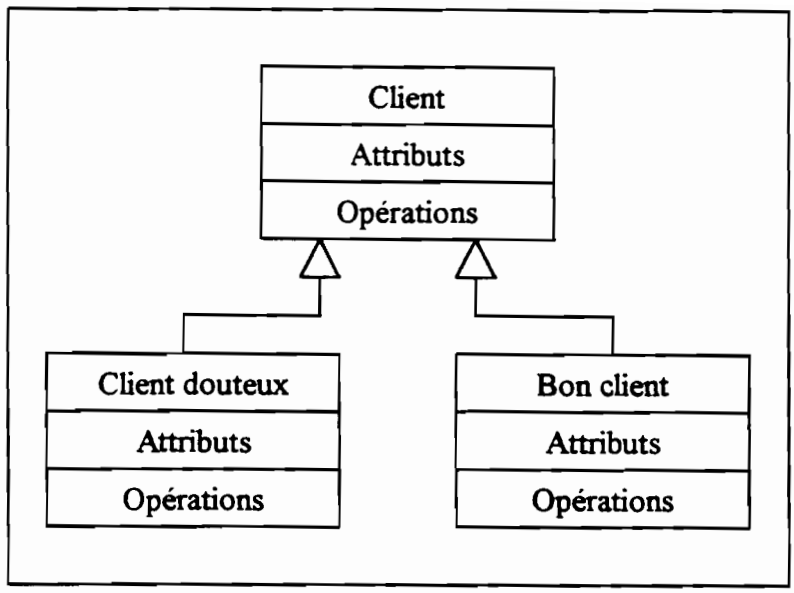


Figure 18 : représentation d'une généralisation

- les diagrammes de cas d'utilisation

Le diagramme de cas d'utilisation permet de décrire le comportement du système du point de vue de l'utilisateur sous forme d'actions et de réactions.

Un cas d'utilisation n'est pas une fonction du système mais englobe plusieurs fonctions. C'est ainsi qu'au lieu de regarder le système de l'intérieur et d'analyser ses fonctions, l'idée consiste à analyser pourquoi l'utilisateur veut utiliser le système.

Trois principaux éléments interviennent dans ce diagramme: les acteurs (personnes ou autres systèmes), les cas d'utilisation et le système. Un cas d'utilisation est représenté par un ovale et la participation d'un acteur à un cas d'utilisation par une flèche simple. Les cas d'utilisation peuvent être en relation (relation d'utilisation ou d'extension).

La description d'un cas d'utilisation peut se faire de plusieurs manières, de la plus informelle (textuelle) à la plus formelle (machines à états).

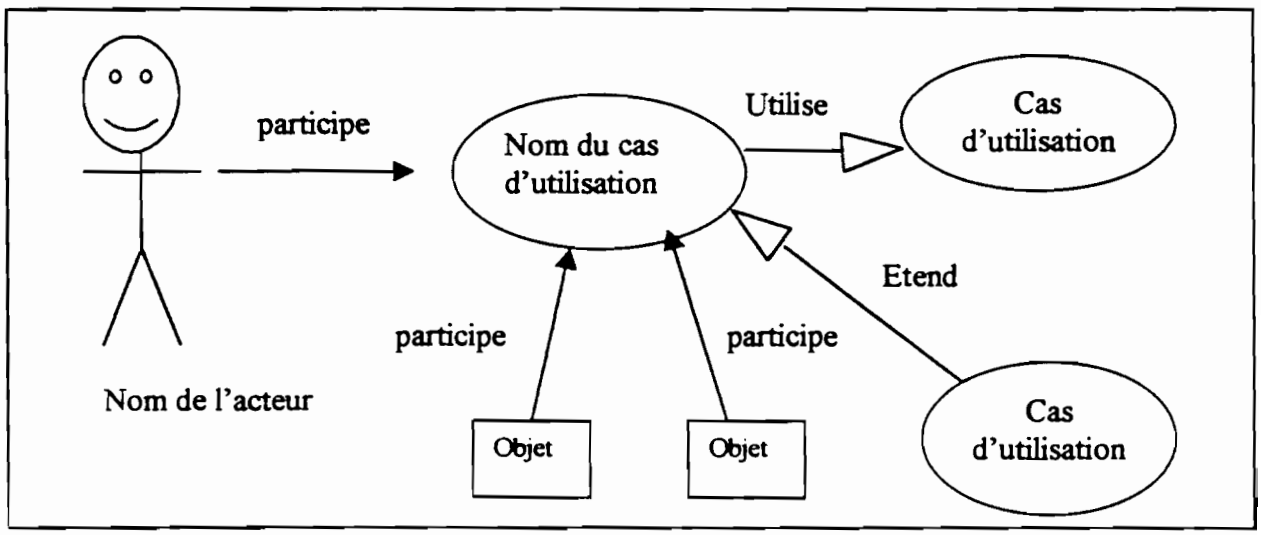


Figure 19 : représentation d'un cas d'utilisation

- *Abstraction*

L'abstraction constitue un des piliers de l'approche orientée objet. On peut en donner deux définitions qui se complètent :

- processus consistant à identifier une entité en mettant en évidence ses caractéristiques pertinentes du point de vue de son utilisation ;
- caractéristiques essentielles d'une entité qui la distinguent de tous les autres types d'entités. Une abstraction définit une frontière relative à la perspective de l'observateur.

- *Encapsulation*

L'encapsulation est un mécanisme qui permet de modéliser en même temps les données et les opérations. Un objet est donc composé d'une interface publique qui contient la spécification des opérations et d'une partie implantation où l'on trouve à la fois la structure de données et la codification des opérations.

- *Surcharge*

La surcharge est une notion générique qui signifie que l'on peut définir plusieurs fois un opérateur de même nom pour des classes ou des types d'objets différents.

- *Composant*

Un composant est un élément binaire qui offre aux autres composants (ses clients) une interface qui leur permet de dialoguer avec le composant en question. Cette interface est un contrat qui définit la manière d'utiliser le composant.

- *Classification / Instanciation*

La classification est une forme d'abstraction par laquelle une collection d'objets de même nature est vue comme un objet unique appelé classe.
L'instanciation est la matérialisation d'une occurrence d'objet d'une classe donnée.

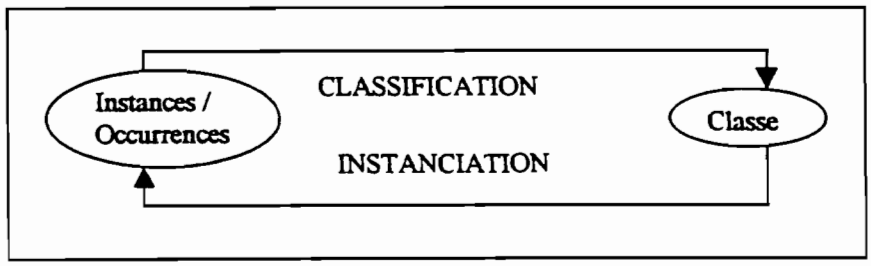


Figure 4 : processus instanciation / classification

- *Notion d'association*

- Une association représente une relation structurelle entre classes d'objets.
- Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées.

- *Notion d'agrégation*

L'agrégation représente une association non symétrique dans laquelle une des extrémités joue le rôle prédominant par rapport à l'autre.

- quelle que soit l'arité, l'agrégation ne concerne qu'un seul rôle d'une association.

(L'arité caractérise le nombre de classes participant à une association et le rôle permet de déterminer la sémantique liée à une association)

- les diagrammes des composants

Les diagrammes de composants permettent la description des éléments physiques et leurs relations dans le système en cours de modélisation. On distingue trois types de composants : la spécification, le corps et la spécification générique.

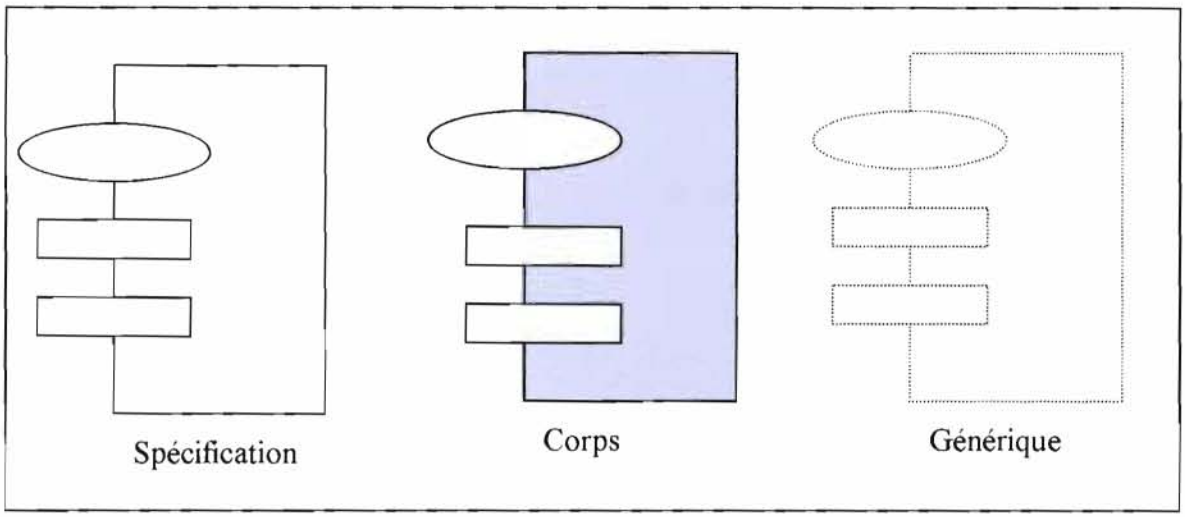


Figure 20 : les différents types de composants

Dépendance entre composants

Les relations de dépendance sont utilisées dans les diagrammes de composants pour indiquer qu'un composant fait référence aux services offerts par un autre composant. Elle est représentée par une flèche en pointillé allant de l'utilisateur vers le fournisseur.

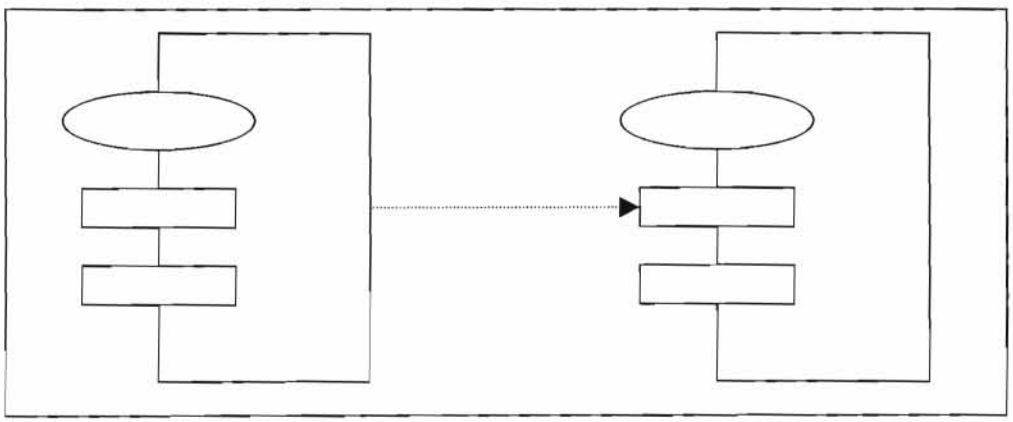


Figure 21 : relation de dépendance entre composants

Tâche
On appelle tâche un composant qui possède son propre flux de contrôle. Les stéréotypes <<Processus>> et <<Flot>> sont prédéfinis dans UML. En UML, les tâches sont représentées de la façon suivante :

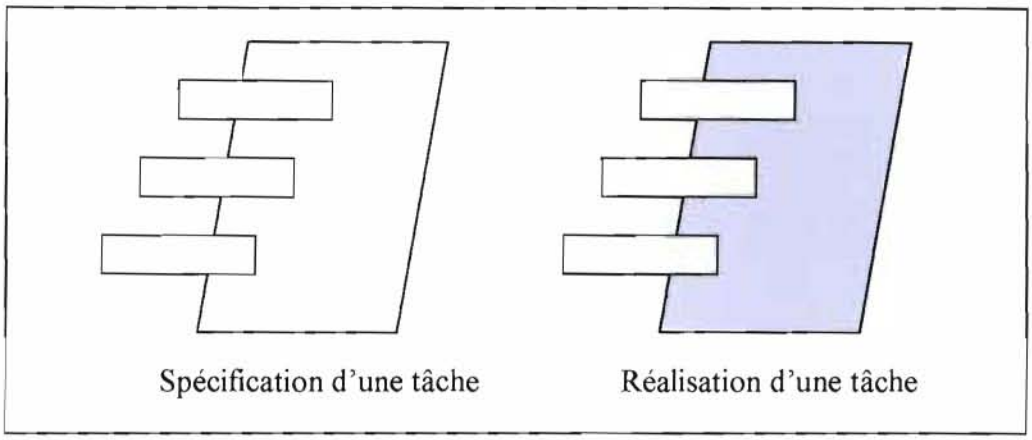


Figure 22 : représentation de tâches

Programme principal / sous-programmes
La notation UML permet de représenter les programmes principaux et les sous-programmes.
Un programme principal représente le point d'entrée d'une application et son nom est souvent utilisé par l'éditeur de liens pour nommer le programme exécutable correspondant à l'application. Ceci permet de relier le modèle des composants avec le modèle des processus. Le programme principal est identifié par une icône.

Un sous-programme regroupe les procédures et les fonctions qui n'appartiennent pas à des classes. Ces composants peuvent contenir des déclarations de types de base nécessaires pour la compilation des sous-programmes, mais ils ne contiennent jamais de classes.

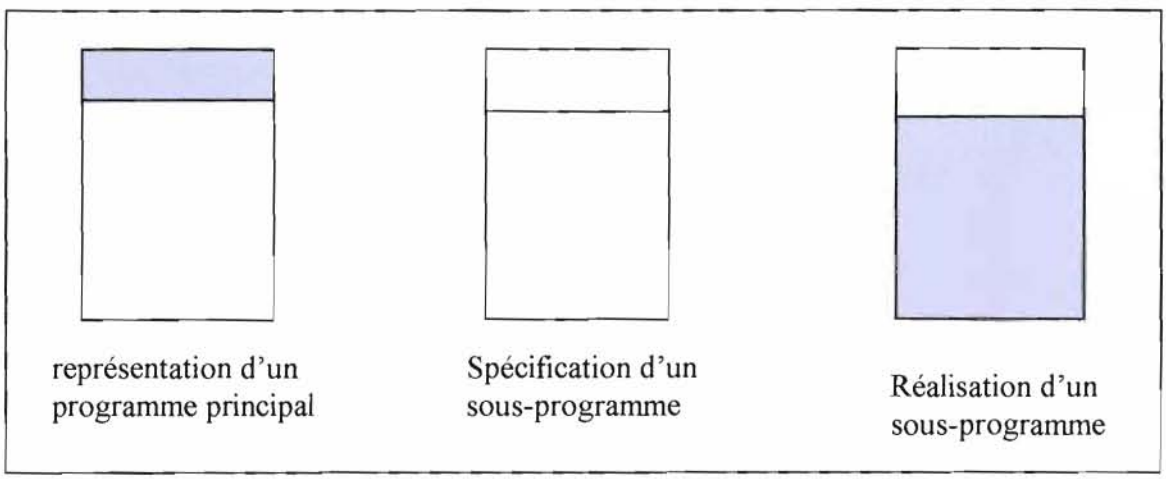


Figure 23 : représentations d'un programme principal et de sous-programmes

Sous-systèmes

Pour faciliter la réalisation des applications, les différents composants peuvent être regroupés dans des paquetages selon un critère logique. Ils sont souvent stéréotypés en sous-systèmes pour ajouter les notions de bibliothèques de compilation et de gestion de configuration à la sémantique de parition déjà véhiculée par les paquetages.

Chaque sous-système peut contenir des composants et d'autres sous-systèmes.

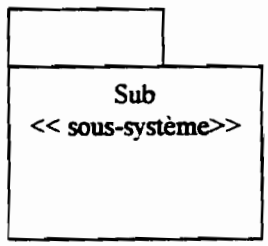


Figure 24 : regroupement de sous-système en paquetage

La décomposition en sous-systèmes n'est pas une décomposition fonctionnelle. Les fonctions du système sont expérimentées du point de vue de l'utilisateur dans la vue des cas d'utilisation.

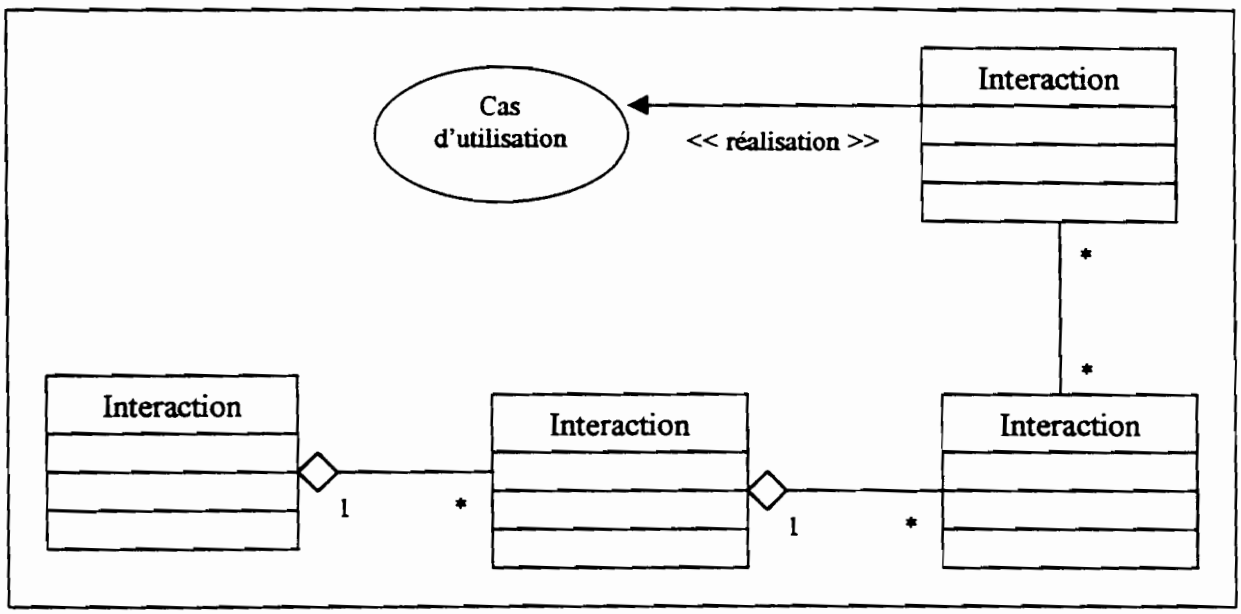


Figure 25 : décomposition en sous-système et décomposition

Outils

- Visual Basic 6.0 Edition Professionnelle

Pour le développement des différents modules, Visual Basic *Edition professionnelle* a été proposé comme outil de développement parce que ce logiciel fait partie des outils disponibles dans l'organisme d'accueil.

Visual Basic est un logiciel de développement qui existe en trois éditions : *l'édition standard*, *l'édition professionnelle* et *l'édition entreprise*. L'édition professionnelle se distingue de l'édition standard principalement par la présence de contrôles et d'outils de développment supplémentaires, et d'un support plus avancé des bases de données. *L'édition entreprise* permet de réaliser des applications OLE (Object Linking Embedding) de type client/serveur.

- Microsoft SQL Server 7.0

Microsoft SQL Severr 7.0 s'est imposé chez les responsables du projet comme système de gestion de base de données. Cela est dû au fait que l'un des objectifs des responsables était la réalisation d'une solution qui nécessite le moins de compétences spécifiques en informatique pour pouvoir assurer le bon fonctionnement de l'application.
Il noter aussi que l'adoption de Microsoft SQL Server ne nécessite pas de compétences très spécifiques dans la gestion des applications qui seront implémentées :

➢ la croissance automatique des bases de données

Par défaut sous Microsoft SQL Server, les bases de données évoluent automatiquement en taille sans autre limite que la taille du disque dur sauf si on définit volontairement la taille de la base de données.

➢ l'administration simpliée

La tâche d'administration sous Microsoft SQL Server est relativement aisée du fait de l'existence de plusieurs assistants. Il faut aussi noter que la plupart des paramètres sont calculés de manière automatique et dynamique.

➢ le nouveau processeur de requêtes

Quelques caractéristiques sont à noter :
- l'utilisation de plusieurs index par table dans une requête;
- le choix de plusieurs stratégies de jointures : itération imbriquée, fusion ou hachage;
- le choix de plusieurs stratégies de regroupement;
- l'exécution des requêtes sur tous les processeurs disponibles, en parallèle;
- statistiques de distribution des données fondées sur des algorithmes d'échantillonnage et mises à jour automatiquement.

Associé à l'analyseur de requêtes et à son plan d'exécution graphique, il s'agit de l'un des meilleurs processeurs heuristiques du marché.

Aussi Microsoft SQL Server intègre un outil qui permet le traitement de données orienté décisionnel. Un système décisionnel est conçu pour répondre à des requêtes complexes en lecture seule (consulation). Un tel système est appelé OLAP (On Line Analytic Processing) et utilise une approche de DataWarehouse permettant de faire certains traitements d'agrégation de données qui seront diffusées.

En fait le DataWarehouse désigne une technique de création de base de données orientée vers le stockage de données en masse. Un datawarehouse est donc constitué d'une ou plusieurs bases de données dénormalisées, fortement indexées, dont les données sont le résultat d'un traitement de modification, d'agrégation, de filtrage, etc.
OLAP est un regroupement de technologies qui transforment des données stockées dans un datawarehouse en cubes multidimensionnels.
Un système OLAP organise les données sous forme "préagrégée", fortement redondantes.
Il est beaucoup plus volumineux et son but étant l'interrogation, la ventilation et le regroupement de données et est organisé pour accélerer au maximum les interrogations.

- Microsoft Access (97 / 2000)

Microsoft Access (97 / 2000) est le SGBD orienté bureautique utilisé pour l'implémentation des différentes applications existantes.

III) APPROCHE DE SOLUTION

III.1) Choix architectural

La problématique liée à la migration des données est un sujet très délicat car les données résultantes n'ont de valeur que si elles sont justes et conformes voire équivalentes aux données de départ.
Il s'agit ici de s'assurer de la présence de toutes les données pertinentes et de leur cohérence d'ensemble.
Pour la migration des données deux cas de figures se présentent en terme d'approche de solution.

Cas 1 :
Dans ce cas il s'agit de partir d'une base de données Access contenant des données pour obtenir une base de données SQL Server équivalente. Pour ce faire l'outil de migration doit créer à l'identique une base de données sous l'environnement SQL Server qui sera capable de contenir les données à transférer.
Ensuite on procède au transfert des données dans les tables correspondantes. Comme nous sommes dans une optique de migration bidirectionnelle de données, l'approche est la même dans les deux sens.
Il faut noter que cette approche de solution nécessite un travail fastidieux et présente un grand risque d'incohérence au niveau de certaines contraintes de données (intégrité référentielle, ...).

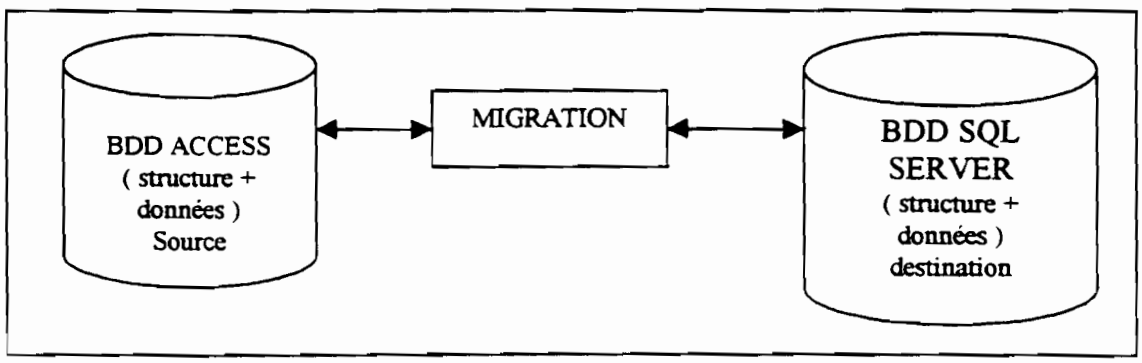


Figure 26 : représentation du cas1

Cas 2 :

Dans cette approche de solution on considère la base de données Access (structure + données) comme source de données et une structure de base de données SQL SERVER destination qui n'est pas forcément identique en terme de définition de schéma à la base de données source. Mais on doit avoir au moins une inclusion de schéma entre les tables sources et les tables destinations.

C'est ainsi qu'après la définition de l'espace de travail de l'application c'est à dire la définition de la correspondance entre les tables sources et les tables destinations, puis des champs sources et des champs destinations, l'outil procède au transfert des données des champs sources vers les champs destinations.

Le même schéma s'applique dans le sens du transfert de données de SQL Server vers Access.

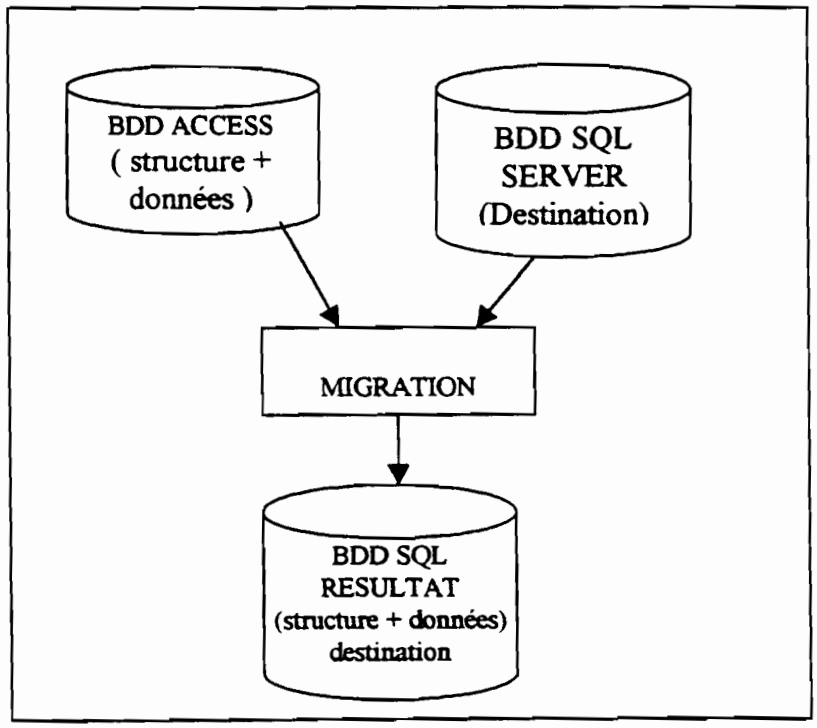


Figure 27 : représentation du cas 2

Précautions à prendre

Il est souvent nécessaire de déplacer des données entre différents environnements ; ce qui peut nécessiter la transformation de données. Avant toute opération d'importation , d'exportation ou de migration il faut :

- identifier la source des données ;
- spécifier la destination des données ;
- transformer les données.

Les opérations de transformation peuvent être simples ou complexes. Lors de la transformation de données on peut être amené à faire :

- de la modification de format des données ;
- de la transformation et du "mappage" des données ;
- du rétablissement de la cohérence des données ;
- de la validation des données (vérification de la précision et de l'exactitude des données).

III.2) Spécification technique du problème

Plusieurs approches de solutions sont envisageables pour la réalisation d'un outil de migration bidirectionnelle des données de SQL Server vers Access, parmi lesquelles nous retiendrons trois d'entre elles.

Solution n°1 :

Cette solution consiste à utiliser un ensemble d'instructions Transact-SQL comme ***select into, insert, select, bulk insert, backup / restore*** et de procédures stockées systèmes telles que ***sp_attach_db*** et ***sp_detach_db*** pour réaliser les opérations de transformation et de migration de données.

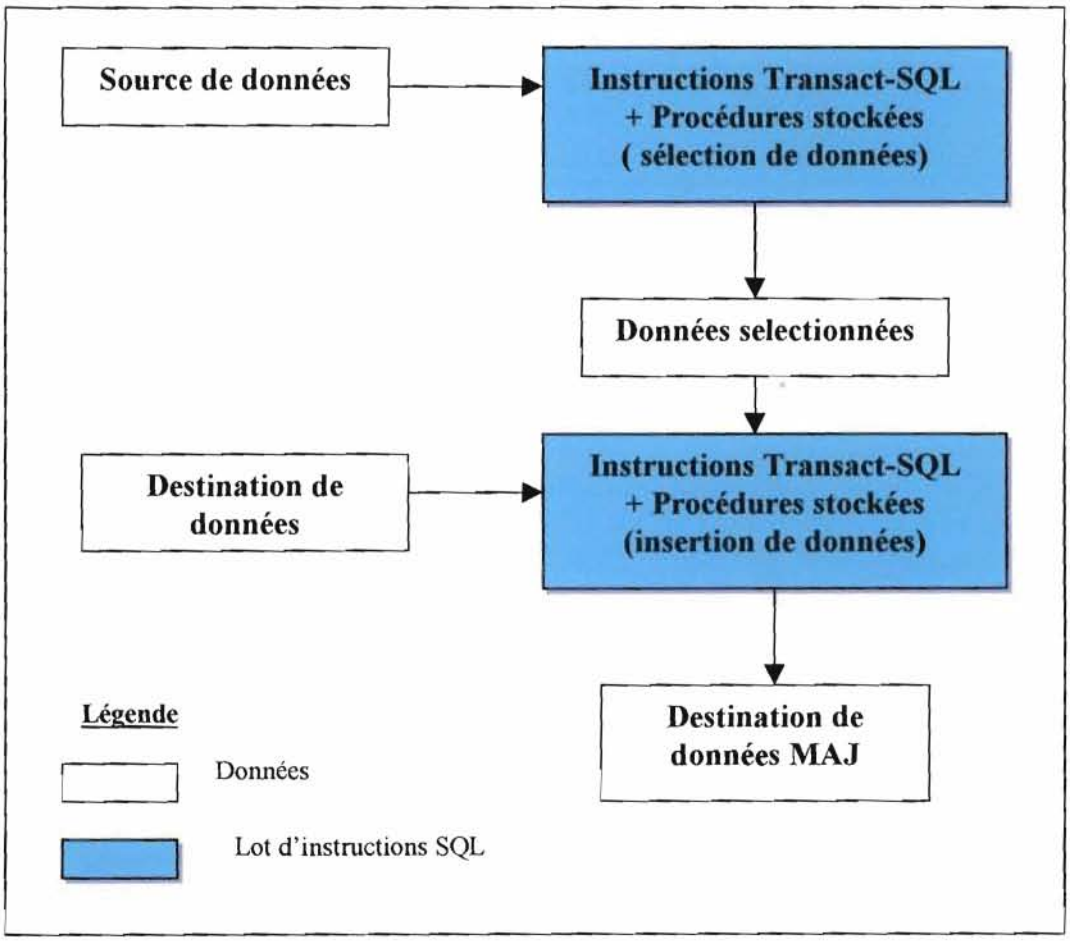


Figure 28: Processus de transfert de données avec des instructions SQL

Solution n°2 :

Cette solution est réalisable grâce aux modèles de programmation et aux interfaces de programmation d'application (API, Application Programming Interface) disponibles dans SQL Server, tel que le modèle d'objets DTS (Data Transformation Service) en se basant sur les services de DTS. Les services de DTS (Distribution Transformation Services) offrent les possibilités de manipulation de données :

- l'importation et l'exportation de données d'une source OLE-DB depuis / vers une autre source OLE-DB;
- la planification de ces importations / exportations;
- la transformation, le filtrage et la normalisation des données au moment de l'importation / exportation via un langage de programmation simple et puissant comme Visual Basic ou Java;
- la mise en place aisée des mises à jour incrémentales;
- la conception graphique de scripts complexes avec le concepteur de lots DTS;

l'utilisation simple des lots DTS puisqu'il s'agit d'objets COM.

A partir de ce modèle d'objets on peut implémenter nos propres programmes pour effectuer les transformations ou la migration de données. En effet les services DTS permettent de :

- créer des objets de transformation personnalisés ;
- développer des applications accessibles par le biais de fournisseurs OLE DB (Object Linking Embedding Database).

Le processus de transformation ou d'importation et d'exportation des données est intégré à tous les systèmes de gestion de base de données . Les services DTS utilisent un fournisseur OLE DB pour transformer et transférer les données à partir d'une ou plusieurs sources et les exporter vers une ou plusieurs destinations, la source et la destination peuvent être hétérogènes. (cf. figure 29). OLE DB prend en charge l'accès à tout format de stockage de données, relationnel ou non , pour lequel un fournisseur OLE DB est disponible. Chaque fournisseur OLE DB est spécifique à un mécanisme de stockage particulier, tel que les bases de données SQL Server , les bases de données Microsoft ACCESS.

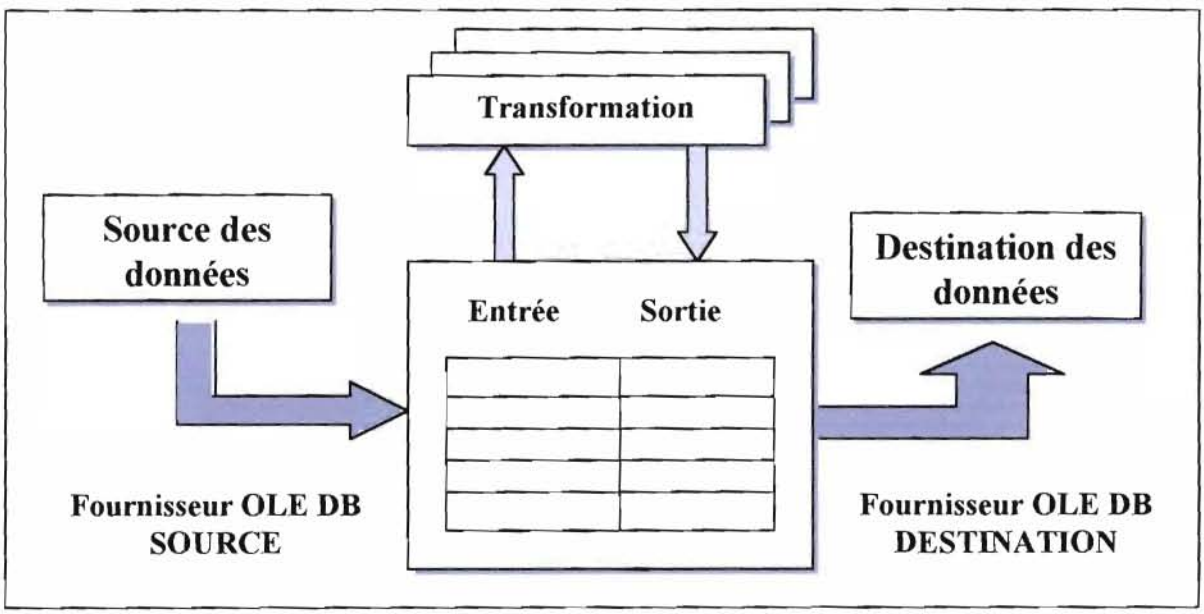


Figure 29 : Processus des services DTS

Le service DTS implique l'accomplissement d'une série de tâches :

1- Création d'un lot DTS ;
2- Transformation et mappage des données ;
3- Définition des tâches de transformation ;
4- Définition des flux de travail ;
5- Définition du lignage des données ;
6- Exécution et planification d'un lot DTS.

Solution n°3 :

Cette approche de solution est basée sur XML (eXtensible Markup Language) qui offre une possibilité d'échange de données. En effet contrairement à HTML (HyperText Markup Language) qui est un langage dédié à la présentation des données, XML est un langage qui permet de manipuler les structures et les contenus des données dans une seule entité. La qualité qu'offre XML de pourvoir intégrer le schéma et les données dans une entité facilitera l'échange des données.

A partir de XML on peut instancier un langage particulier de marqueurs permettant de représenter les données et les structures des différentes bases de données. L'instanciation d'un langage spécifique à partir de XML se traduit par la définition d'un type de document DTD (Document Type Definition) qui est en réalité la définition du langage de marqueurs (ensemble de balises de début et de fin).

Dans notre cas spécifique de migration bidirectionnelle de données de ACCESS à SQL SERVER il sera associé à chaque table (schéma + données) une DTD particulière qui décrit la structure de la table.

C'est ainsi qu'en utilisant une notation avec une approche objet, une DTD d'une base de données sera décrite selon la structuration de la **figure 30** .

L'architecture globale de la migration est décrite au niveau la **figure 31** .

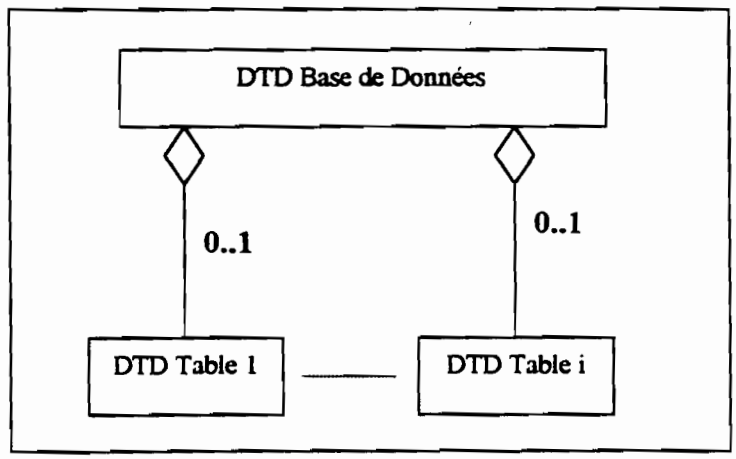


Figure 30 : Structure d'une DTD d'une base de données

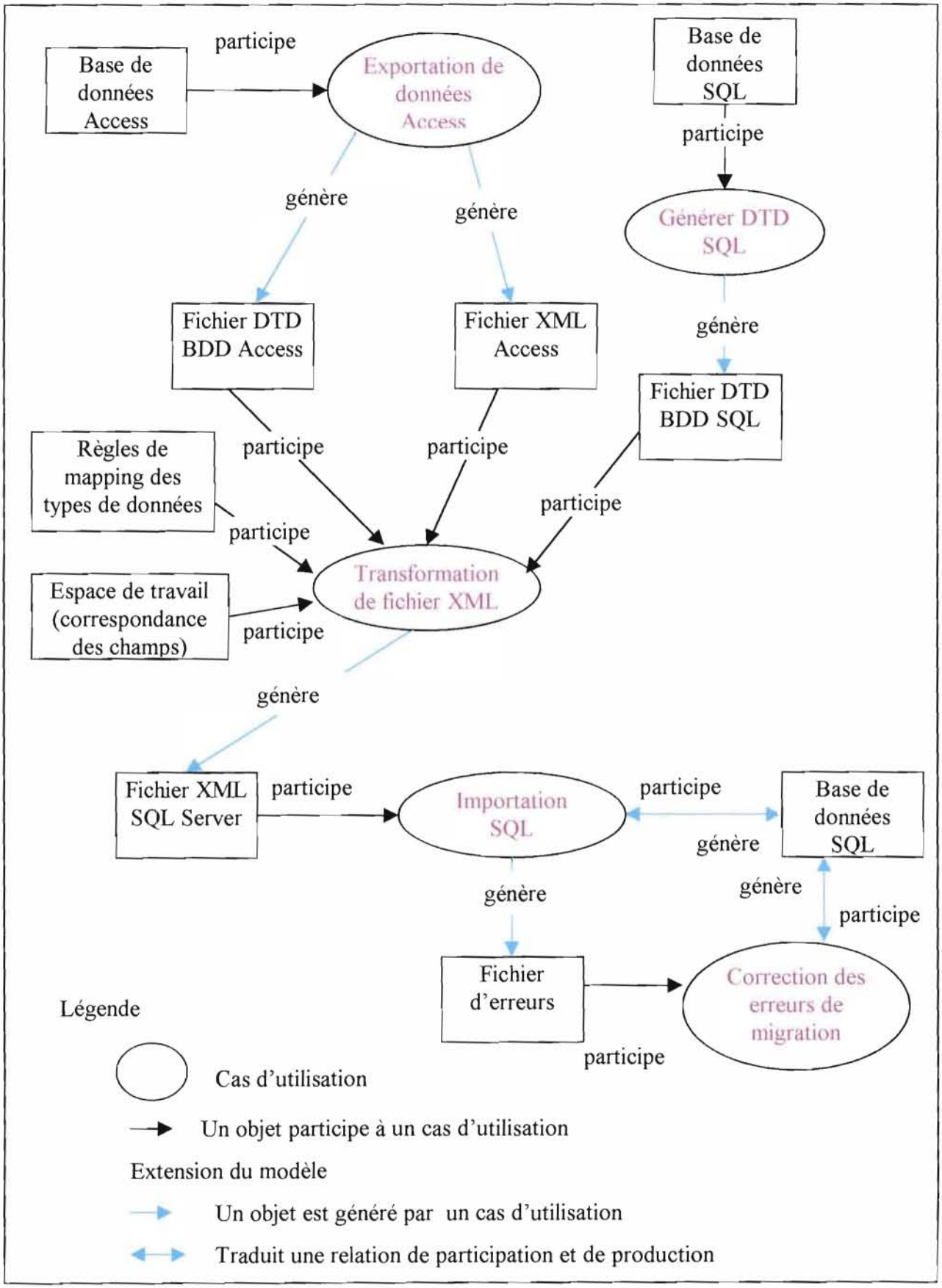


Figure 31.a : Architecture globale de la migration avec les cas d'utilisation

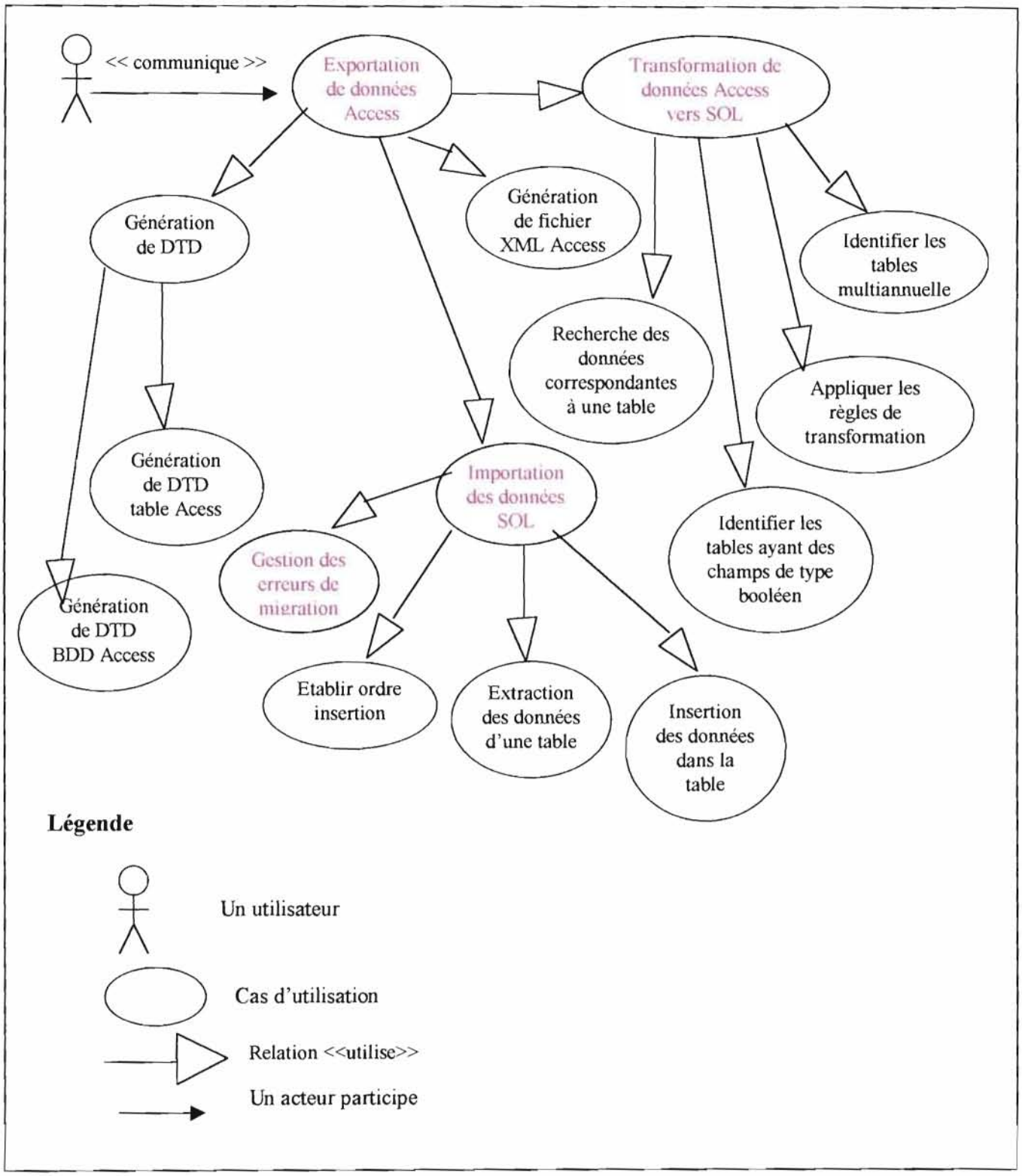


Figure 31.b : Architecture globale de migration avec des cas d'utilisation

Critiques :

Avantages :

- L'approche XML offre une flexibilité au niveau de l'évolution des schémas de chaque base de données car il suffit de modifier la DTD de la base de données concernée par l'évolution de schéma ;
- Elle permet aussi d'éviter de tomber dans des solutions propriétaires.

Inconvénients :

- Cette approche pourrait engendrer un problème de gestion d'environnement de développement dans le cas où les différents traitements ne sont pas implémentés dans le même langage.

III.3) Solution conceptuelle

III.3.1) Conception de la base de données SQL Server

Définition d'une base de données

Selon Marc Humbert dans son ouvrage "les bases de données" paru aux éditions HERMES, "une base de données est un ensemble structuré de données reliées entre elles et stockées de manière cohérente, sans redondance inutile, de sorte qu'elles puissent être manipulées d'une manière efficace par plusieurs utilisateurs concurrents".

Problématique

Il s'agit de concevoir une structure de données capable de mémoriser les données statistiques sur l'enseignement de base afin de permettre le bon fonctionnement des différents frontaux (interfaces) qui seront développés. Parmi les données à mémoriser il existe des données très instables telles que la répartition administrative. Cette instabilité dans le découpage administratif pose un énorme problème dans la mesure où l'on voudrait trouver une structure de données capable de permettre des traitements sur l'évolution des indicateurs du sytème éducatif sur plusieurs années.
Pour ce faire plusieurs approches de conception de la structure de données seront proposées.

Scénario 1

Dans ce scénario nous considérons que les données de chaque année constituent une base de données autonome. Dans chaque base de données on y trouve les données enquêtes de l'année, en plus des données relatives à la répartition géographique des écoles (ATLAS) et celles relatives aux nomenclatures.

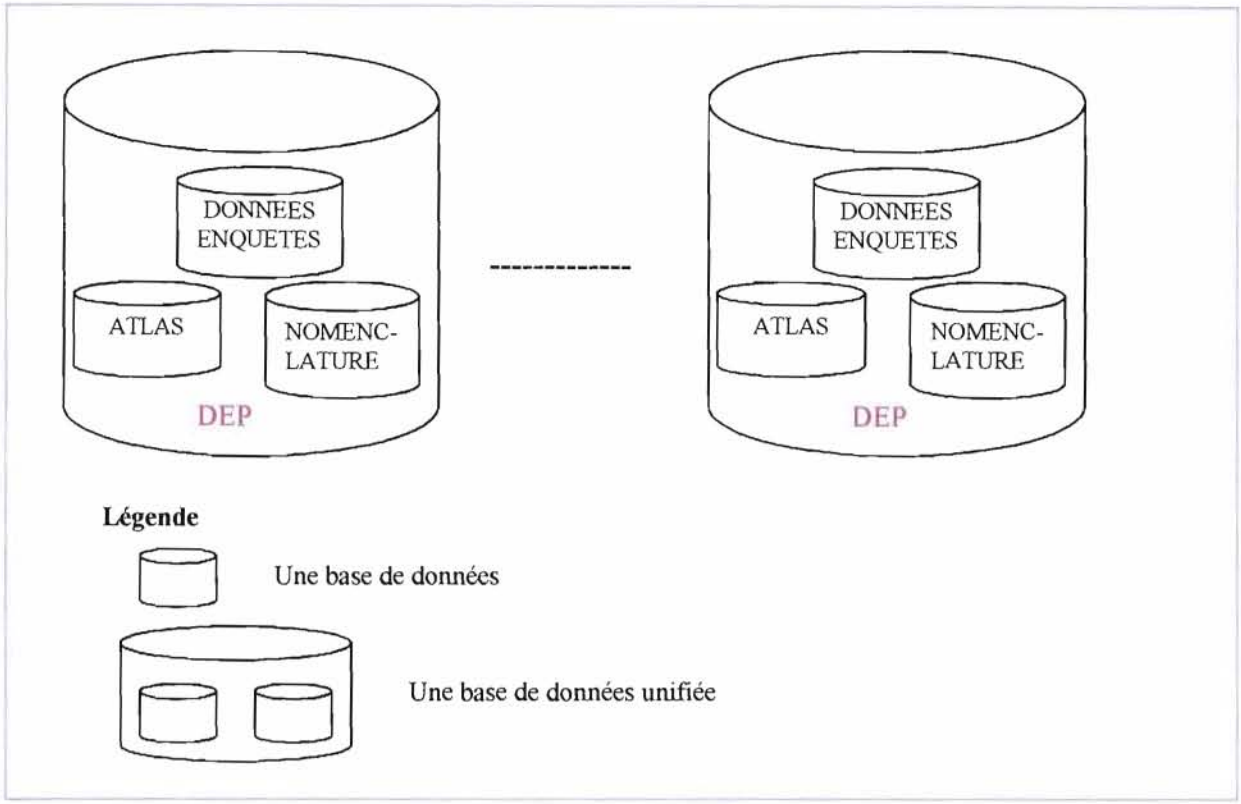


Figure 32 : Structuration des données en plusieurs base de données

Avantages
Toutes les données par année sont conservées.

Inconvénients
L'existence de plusieurs bases de données cause un travail énorme pour le traitement de certains indicateurs sur plusieurs années ; car il faut considérer plusieurs bases de données en même temps.

Scénario 2
Dans ce scénario l'on suppose l'existence de trois bases de données :

- ***la base de données Enquête (BDD Enquête) :***

Cette base de données contient toutes les informations relatives aux données enquêtes de toutes les années. Il faut noter que dans cette base de données ne figurent pas les informations relatives à la répartition géographique des écoles (ATLAS) ainsi que celles des nomenclatures.

- ***la base de données Atlas (BDD Atlas)***

Cette base de données contient uniquement les données relatives à la répartition géographique des écoles (ATLAS).

- ***la base de données Nomenclature (BDD Nomenclature)***

La base de données Nomenclature contient uniquement les informations relatives aux nomenclatures.

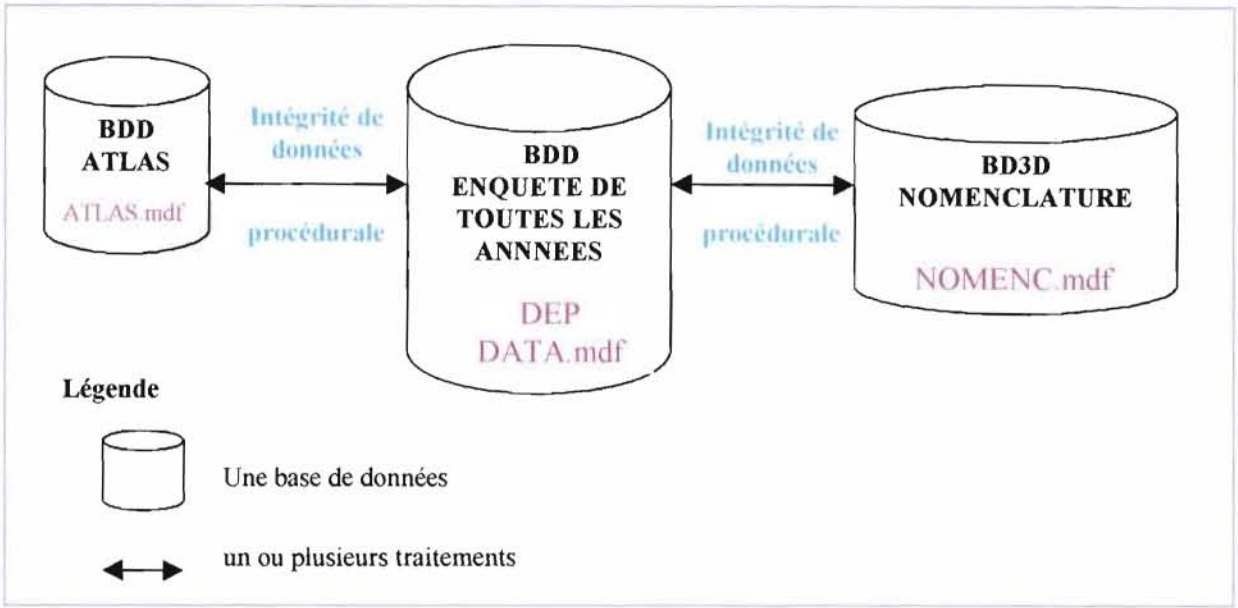


Figure 33 : Structuration des données en trois bases de données

Avantage

L'existence de toutes les données enquêtes de toutes les années dans une seule base de données facilite le traitement sur l'évolution des indicateurs sur plusieurs années.

Inconvénients

L'existence de trois bases de données pourrait engendrer une difficulté dans la maintenance de la cohérence des données du fait qu'on utilise une intégrité de données procédurale non gérée par le système de gestion de base de données.

Scénario 3

Dans ce scénario on considère que toutes les données de toutes les années sont stockées dans une seule base de données. On trouve aussi dans cette base de données les informations relatives à la répartition géographique des écoles (ATLAS), ainsi que celles relatives aux nomenclatures.

Le problème de l'historisation des données au niveau de certaines tables pourrait se faire par l'introduction d'un champ date (l'année) au niveau des clés.

Pour des raisons d'optimisation il sera possible d'implémenter des vues pour chaque année. Ces vues favoriseraient la réplique de la base de données SQL vers une base de données équivalente en Access pour une année donnée.

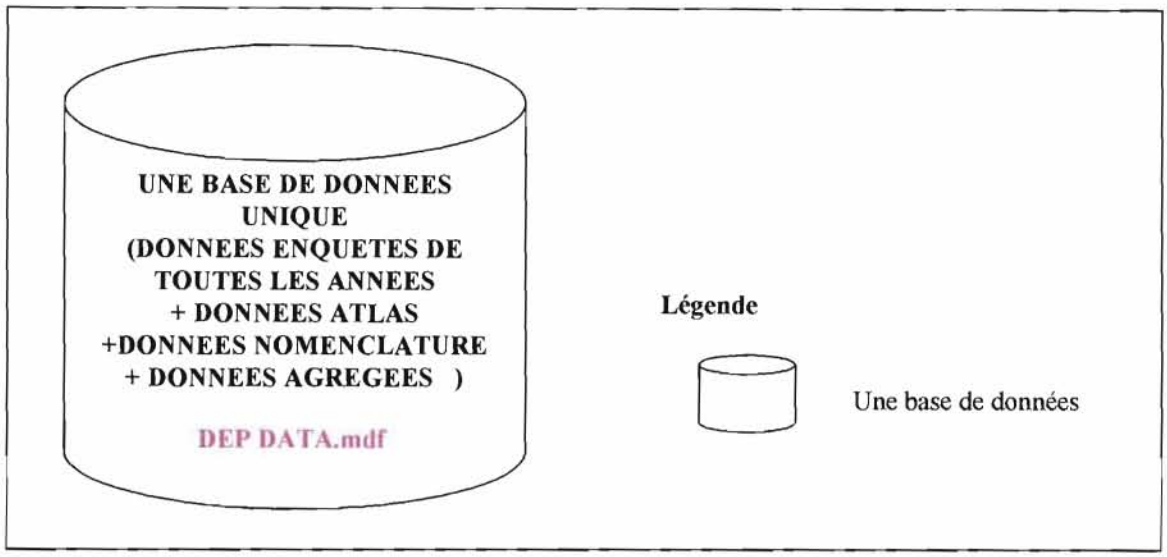


Figure 34 : structuration des données en une base de données

Avantages

L'existence de toutes les données dans une seule base de données.La conservation de l'historique des données est un atout pour le traitement de l'évolution des indicateurs sur plusieurs années du fait de l'existence de toutes les données dans une seule base de données.

Inconvénients

Ce scénario a pour inconvénient majeur d'avoir à stocker dans une seule base de données un volume important de données, ce qui pourrait être source de ralentissement au niveau de certains traitements.

Remarque :

Le scénario choisi sera presenté par la suite.

II.3.2) Politique de sauvegarde et de reprise sur panne

Stratégie de sauvegarde

Assurer la sécurité de fonctionnement de la base de données est une tâche essentielle. C'est ainsi qu'il faut instaurer des stratégies et des procédures garantissant le retour à la normale en cas d'incident logiciel ou matériel.

L'opération de sauvegarde consiste à copier la base de données dans un endroit sûr, typiquement sur bande ou sur un autre disque.

Ainsi notre stratégie de sauvegarde consistera par une sauvegarde complète de la base ou des bases de données, puis pratiquer des sauvegardes incrémentales regulièrement dont la périodicté est annuelle du fait que les données n'évoluent qu'annuellement. Les différentes sauvegardes peuvent se faire par :

- le système d'exploitation
  Elle consitera à copier tous les fichiers de la base de données dans un endroit sûr. Il est impératif de copier tous les fichiers se rapportant à la base de données.
- Le système de gestion de base de données
  La plupart des systèmes de gestion de base de données intègrent des outils permettant d'effectuer des sauvegardes.

Stratégie de reprise sur panne

Une procédure de reprise est une procédure exécutée lors du redémarrage du système ayant pour objectif de reconstruire une base de données cohérente aussi proche que possible de l'état atteint lors de la panne ou de l'arrêt .
En général ces procédures consistent à repartir à partir du journal, et éventuellement de sauvegardes, pour reconstituer une base cohérente en perdant le minimum du travail exécuté avant l'arrêt.

III.3.3) Présentation du SGBD choisi

Définition et rôle d'un SGBD
Un système de gestion de base de données (SGBD) est un ensemble d'outils logiciels permettant la gestion d'une base de données. Les fonctions essentielles d'un SGBD sont:
- la gestion de la persistantce des données;
- la gestion de la résistance aux pannes;
- la gestion de la concurrence;
- la facilité d'interogation.

Il faut noter que conformément aux contraintes expliquées par les responsables de la structure d'accueil, SQL Server 7.0 a été choisi comme SGBD. SQL Server est un SGBD de type relationnel et tire ses fondements du modèle relationnel.

Le modèle relationnel

Le modèle relationnel fut formalisé par E.F. Codd dans les années 70. Depuis, les règles de Codd sont devenues la référence lors de la définition de la plupart des systèmes de gestion de bases de données.
Le modèle relationnel séduit, car il offre un certain nombre d'avantages encore valables aujourd'hui:
- l'utilisation d'un langage de haut niveau pour accéder à l'information sans connaître sa représentation interne;
- un langage de définition de données (Data Definition Language ou DDL) et un langage de manipulation des données (Data Manipulation Language ou DML), standardisés par l'ANSI;
- la gestion interne de la cohérence des informations, par des mécanismes qui assurent l'intégrité des données et des règles;
- la gestion d'objets binaires complexes ou structurés.

III.3.4) Le système d'exploitation choisi

SQL Server ne fonctionnant que sous Windows NT il est évident que le système d'exploitation pour lequel nous optons est Windows NT.

III.4) Solution technique

III.4.1) Choix des composants

Architecture générale des composants disponibles

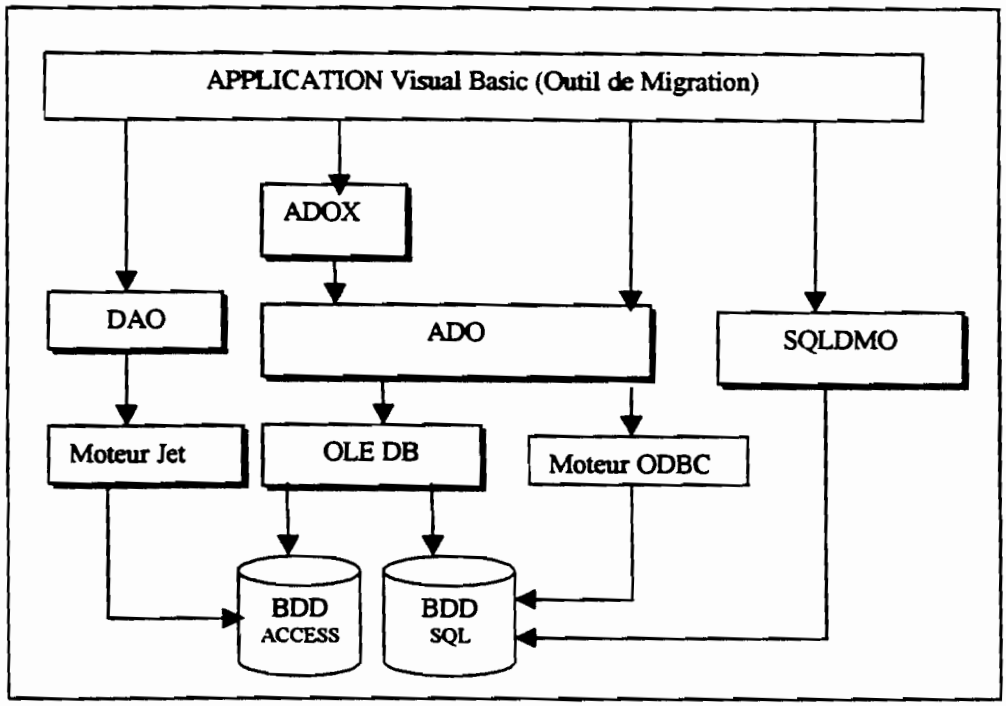


Figure 35 : architecture des composants disponibles

Data Access Objects (DAO)

DAO reste actuellement la méthode la plus répandue pour accéder aux données et en particulier pour l'accès aux bases de données bureautiques.

Le modèle DAO est basé sur une organisation hiérarchique, avec dans l'ordre d'entrée en scène, le moteur Jet DBEngine, les espaces de travail (workspaces), les bases de données (databases), les requêtes, les tables, les tabledefs, ...

Le vivier de talentueux programmeurs DAO est énorme, mais les jours de cette technologie de pointe sont probablement comptés pour plusieurs raisons qui seront évoquées dans les lignes qui suivent.

ActiveX Data Objects (ADO)

ADO est le sucesseur de DAO : il écrase le modèle objet utilisé par DAO. Cela signifie qu'il contient moins d'objets et plus de propriétés, de méthodes et d'évènements.
La plupart des fonctionnalités de DAO ont été consolidées en objets uniques, fabriqués pour un modèle objet bien plus simple. A cause de cela, certains programmeurs DAO peuvent trouver difficile d'avoir à trouver l'objet, la collection, la propriété, la méthode ou l'événement DAO appropriés. Il faut aussi noter qu'à ce jour ADO ne supporte pas toutes les fonctionnalités de DAO.

En tant que programmeurs Visual Basic la question qui se pose à nous est la suivante :
allons- nous utiliser la méthodologie éprouvée solide, robuste et fiable de DAO ? ou devons-nous penser qu'ADO répresente l'avenir et que nous pourrions aussi bien écrire toutes nos applications pour le futur ?

En fait, la reponse à cette question n'est aussi difficile à trouver:
Si on essaie de mettre au point une solution bureautique, l'approche traditionnelle DAO est celle à adopter. Le modèle ADO est un mode d'accès aux données que Microsoft recommande vivement d'utliser car il est probable que le modèle DAO disparaisse dans les prochaines versions de Visual Basic. ADO est une bibliothèque d'instructions qui permet de gérer de nombreux paramètres pour se connecter à des sources de données celles-ci ne vont plus être définies uniquement par l'intermédiaire d'ODBC, mais avec un fournisseur d'accès aux données (ou OLEDB).
Dans le modèle ADO, quatre objets majeurs sont maintenant à notre disposition: l'objet "connection" qui determine les propriétés de la connexion vers une source de données, l'objet "command" qui indique les propriétés des tables ou des requêtes utilisées, l'objet "recordset" qui determine les propriétés des enregistrements et enfin la collection "Errors" qui propose un certain nombre de méthodes et de propriétés pour répondre aux rares erreurs provoquées par Visual Basic.

ActiveX® Data Objects Extensions (ADOX)

Microsoft® ActiveX® Data Objects Extensions (ADOX) pour le Langage de Définition des Données (DDL = Data Definition Language and Security) est une extension apportée aux objets et au modèle de programmation ADO. ADOX contient des objets pour la création et la modification de schémas, aussi bien que pour la sécurité. Du fait que la manipulation des schémas repose sur une approche à base d'objets, il est possible d'écrire du code qui fonctionnera avec différentes sources de données, quelles que soient les différences dans leur syntaxe d'origine.

ADOX est une bibliothèque auxilliaire du noyau des objets ADO. Elle propose des objets supplémentaires pour créer, modifier, et supprimer des objets de schéma, tels que les tables et les procédures. Elle comprend également des objets de sécurité pour assurer le suivi des utilisateurs et des groupes, ainsi que pour accorder et révoquer les autorisations concernant les objets.

Pour utiliser ADOX avec un outil de développement, il faut établir une référence à la bibliothèque de type ADOX. La description de la bibliothèque ADOX est « Microsoft ADO Ext. for DDL and Security ». Le fichier de la bibliothèque ADOX s'appelle Msadox.dll, et l'ID de programme (ProgID) est « ADOX ».

SQL Distributed Manipulation Object (SQLDMO)
SQLDMO est une API permettant la manipulation des structures dans l'environnement SQL Server.

Critères de choix entre DAO et ADO

- **voici quelques bonnes raisons d'utiliser DAO**

- Si on modifie une application DAO existante, il est préférable de continuer à l'utiliser;
- Si l'on déploie une petite solution bureautique ou fonctionnant sur un serveur local, on ne peut pas se tromper avec DAO;
- L'évolution d'ADO est probablement encore trop récente pour faire migrer la plupart des composants DAO en composants ADO maintenant, car ADO ne supporte pas encore les utilisateurs, les groupes, etc.
- Si la sécurité des bases de données constitue l'un des problèmes, il est souhaitable de rester avec DAO pour le moment;
- Microsoft Access lui-même utilise le moteur Jet DAO;
- Le contrôle de données DAO (qui relie rapidement et facilement les programmes Visual Basic aux données de bases) est un contrôle intrinsèque.

- **Voici quelques bonnes raisons d'utiliser ADO**

Les composants ADO constituent un choix dans certaines situations :

- si un projet n'en est qu'au stade de la conception et que l'on compte déployer des données sur l'Internet ou utiliser une source de données autre qu'Access, il est probable envisager l'utilisation des composants ADO;
- ADO est en fait plus facile pour l'utilisateur DAO;
- ADO est plus puissant que DAO et facilite l'accès à plus de sources de données;
- Si on développe des applications client-serveur en utilisant DAO et qu'on ne se fie pas au moteur Jet, on doit alors probablement migrer dès maintenant vers ADO.
- Introduit avec Visual Basic 6.0, ADO constitue le modèle objet d'accès aux données standard pour tous les outils Microsoft.

Remarque :
Le choisi de la technologie d'accès aux données sera presenté dans la partie implémentation.

III.4.2) Architecture technique

Diagramme des composants de l'application

L'application qui sera réalisée comporte trois rubriques qui sont :

- une rubrique d'aide;
- une rubrique de transformation des données;
- une rubrique de configuration.

Remarque :
Chaque module sera considéré comme un composant du système informatique en cours d'implémentation.

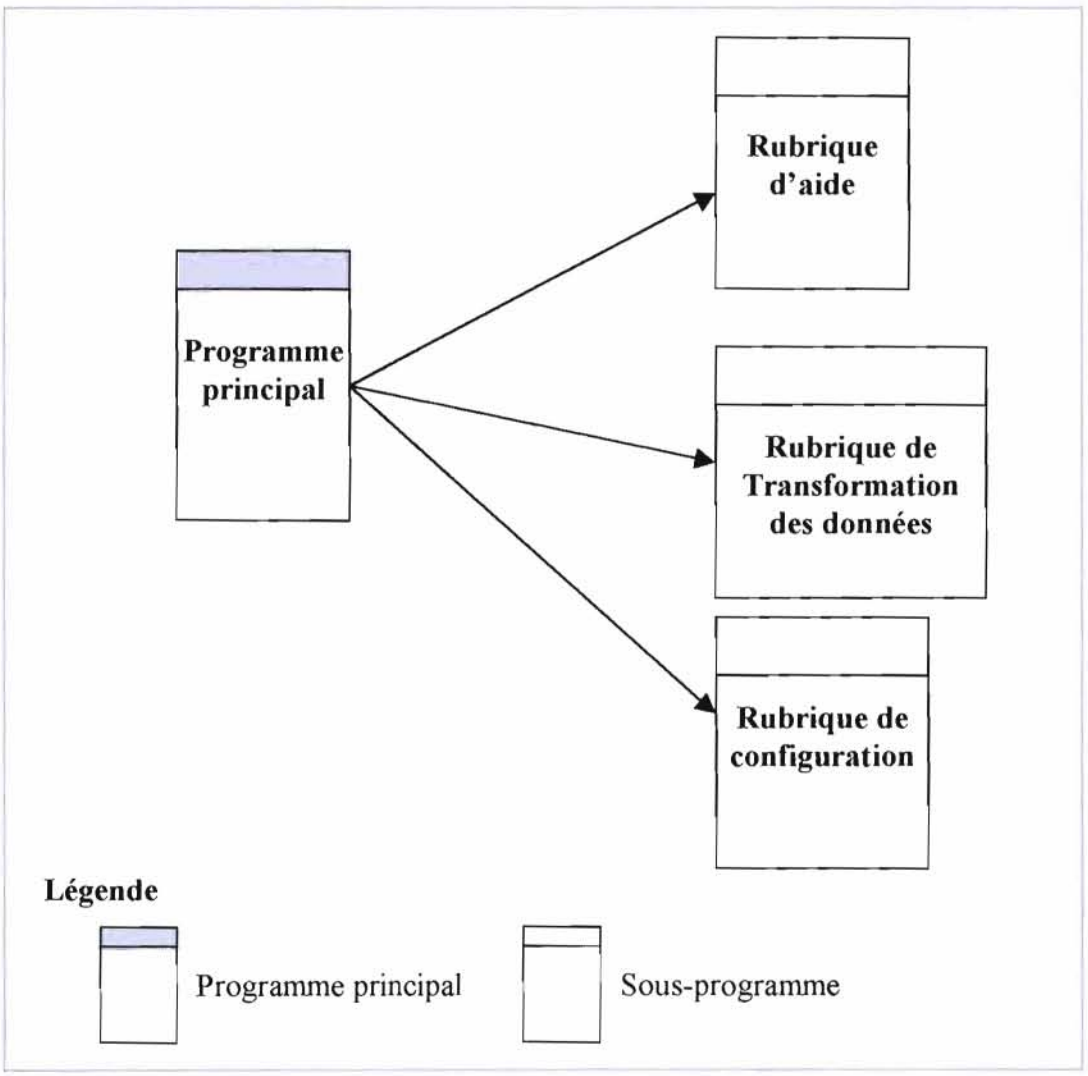


Figure 36 : Architecture globale de l'application

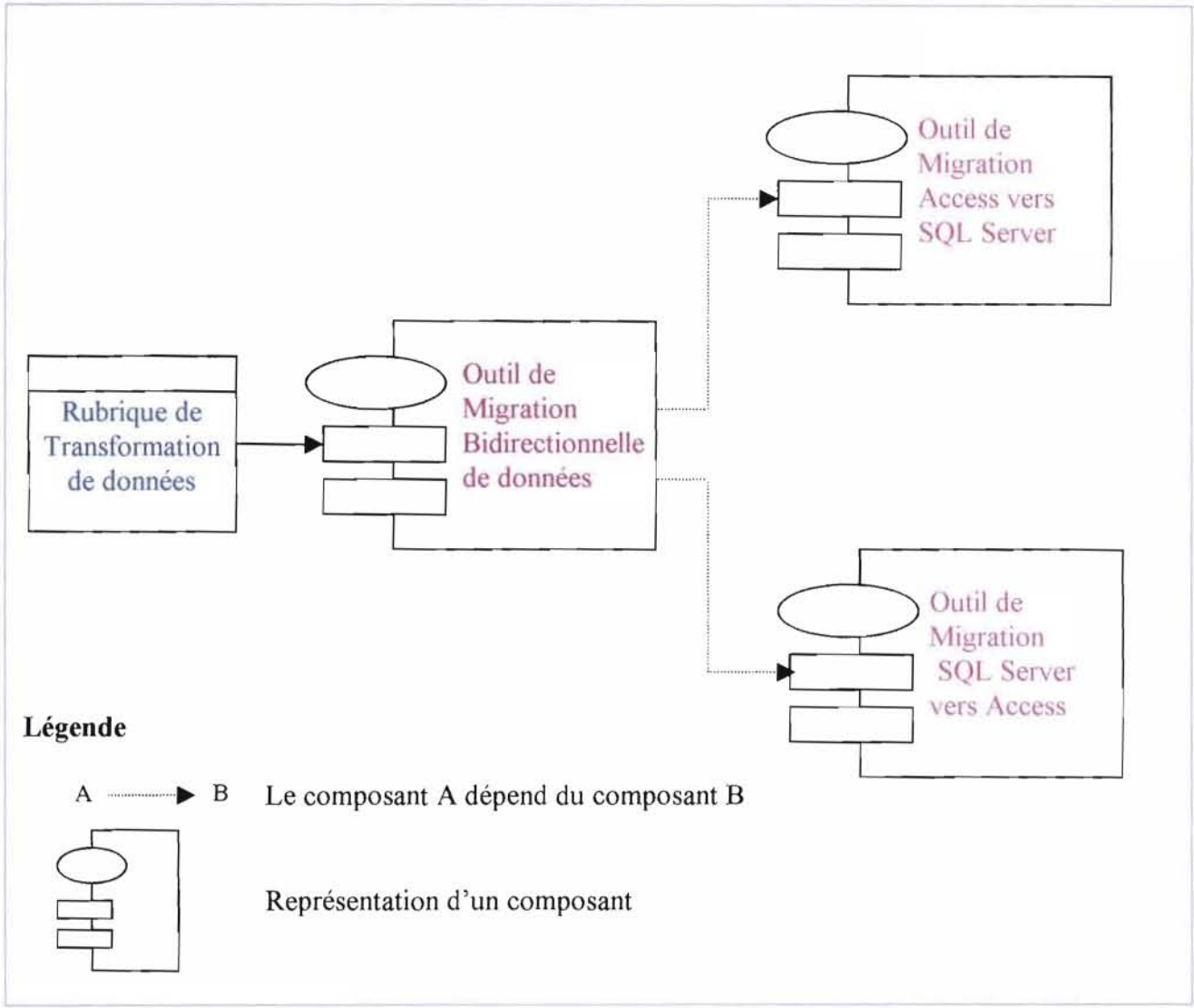


Figure 37 : Décomposition du sous-programme Rubrique de transformation des données

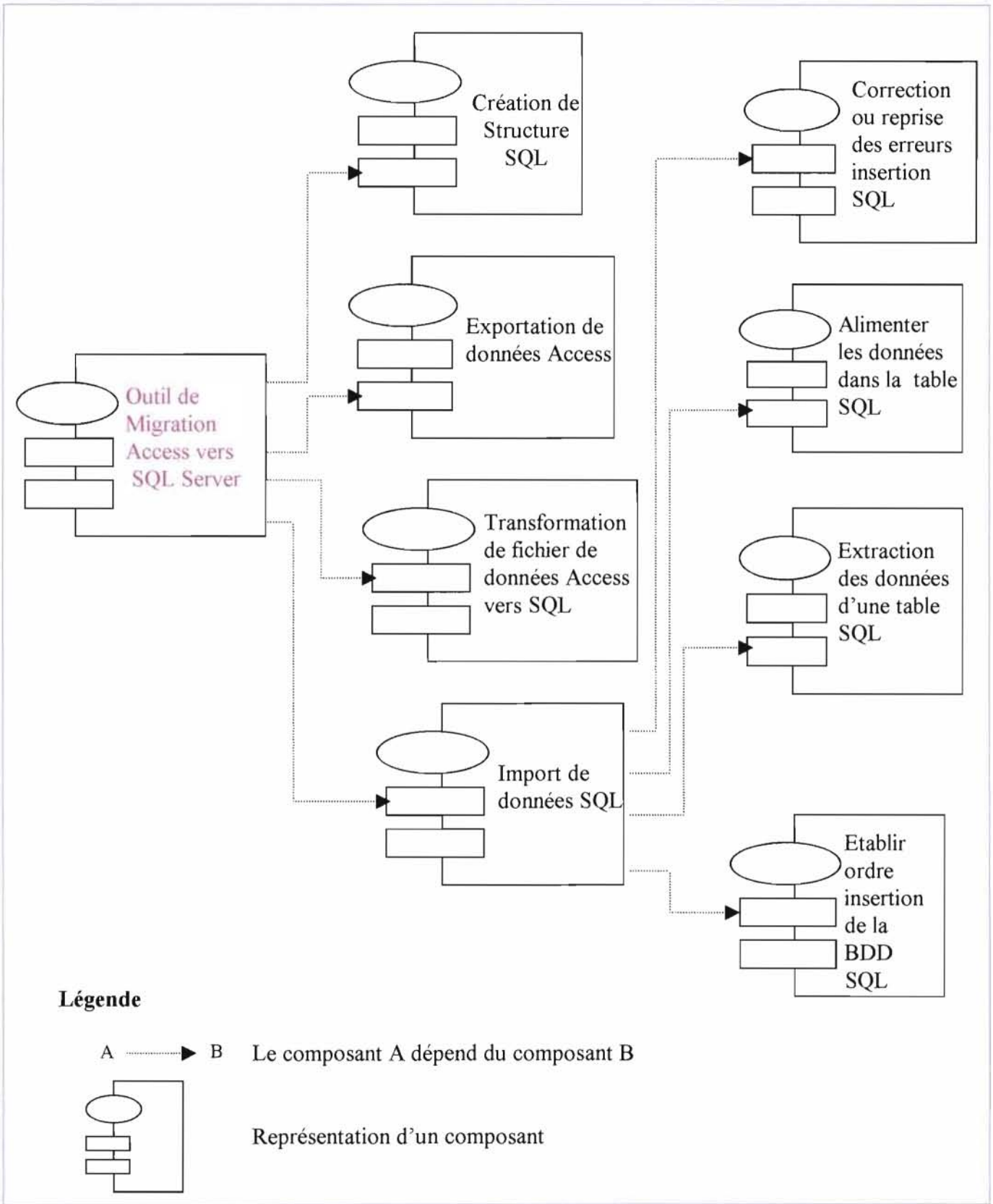


Figure 38 : Décomposition du composant outil de migration Access vers SQL

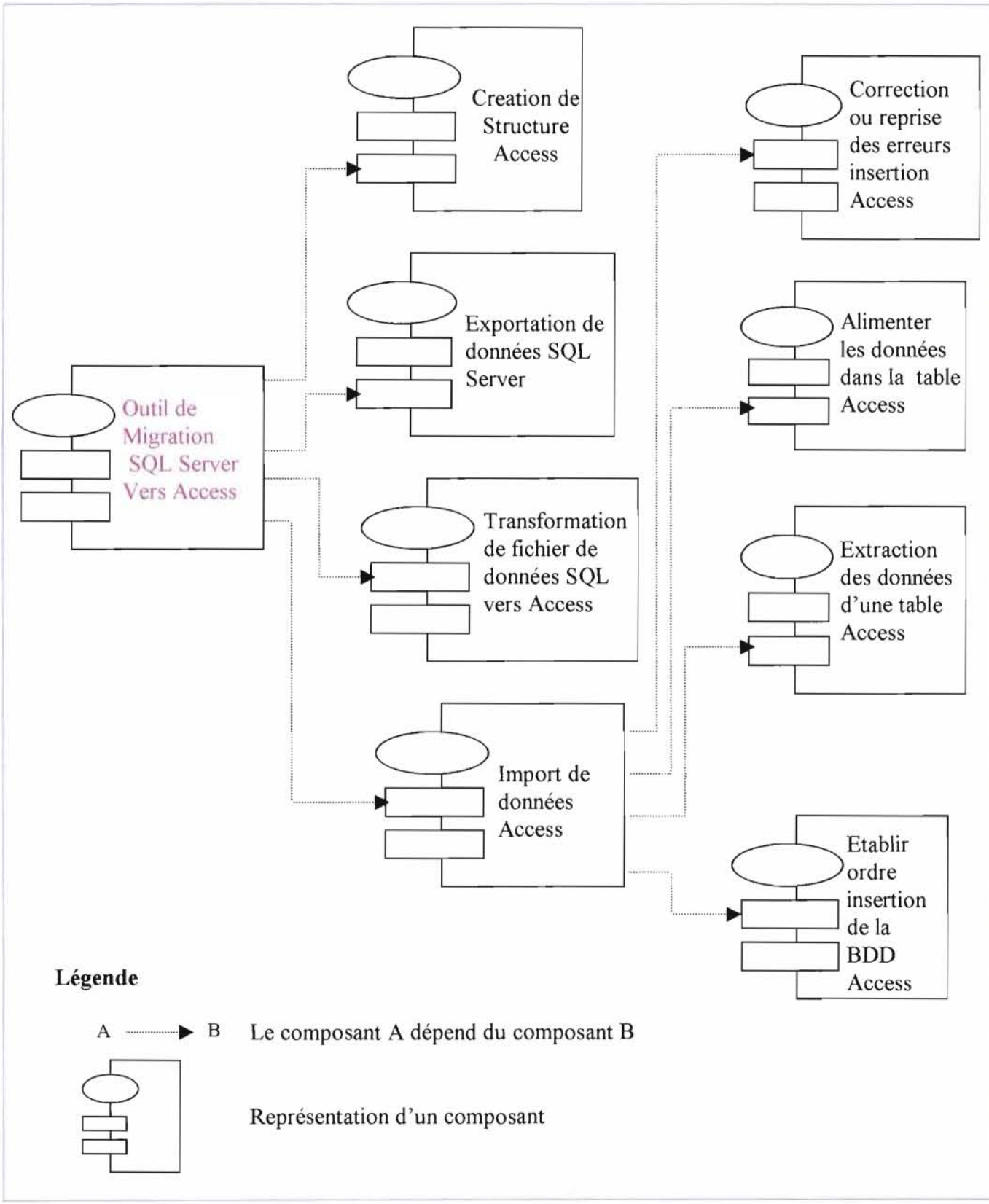


Figure 39 : Décomposition du composant outil de migration SQL vers Access

La notion de module peut être considérée comme un objet composé d'une spécification (interface), d'un corps et d'une description informelle.
Plus généralement le module est considéré comme un composant logiciel qui est souvent composé d'objets simples et peut également être lui-même composant d'un objet plus complexe.

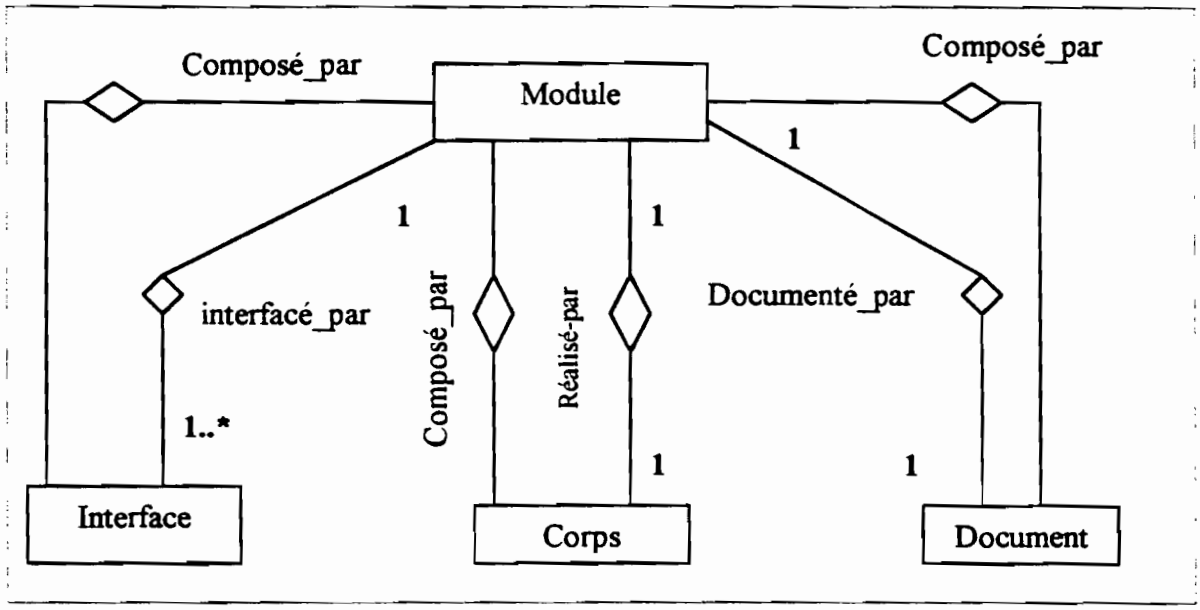


Figure 40 : composition d'un module

Diagramme de dépendances des modules

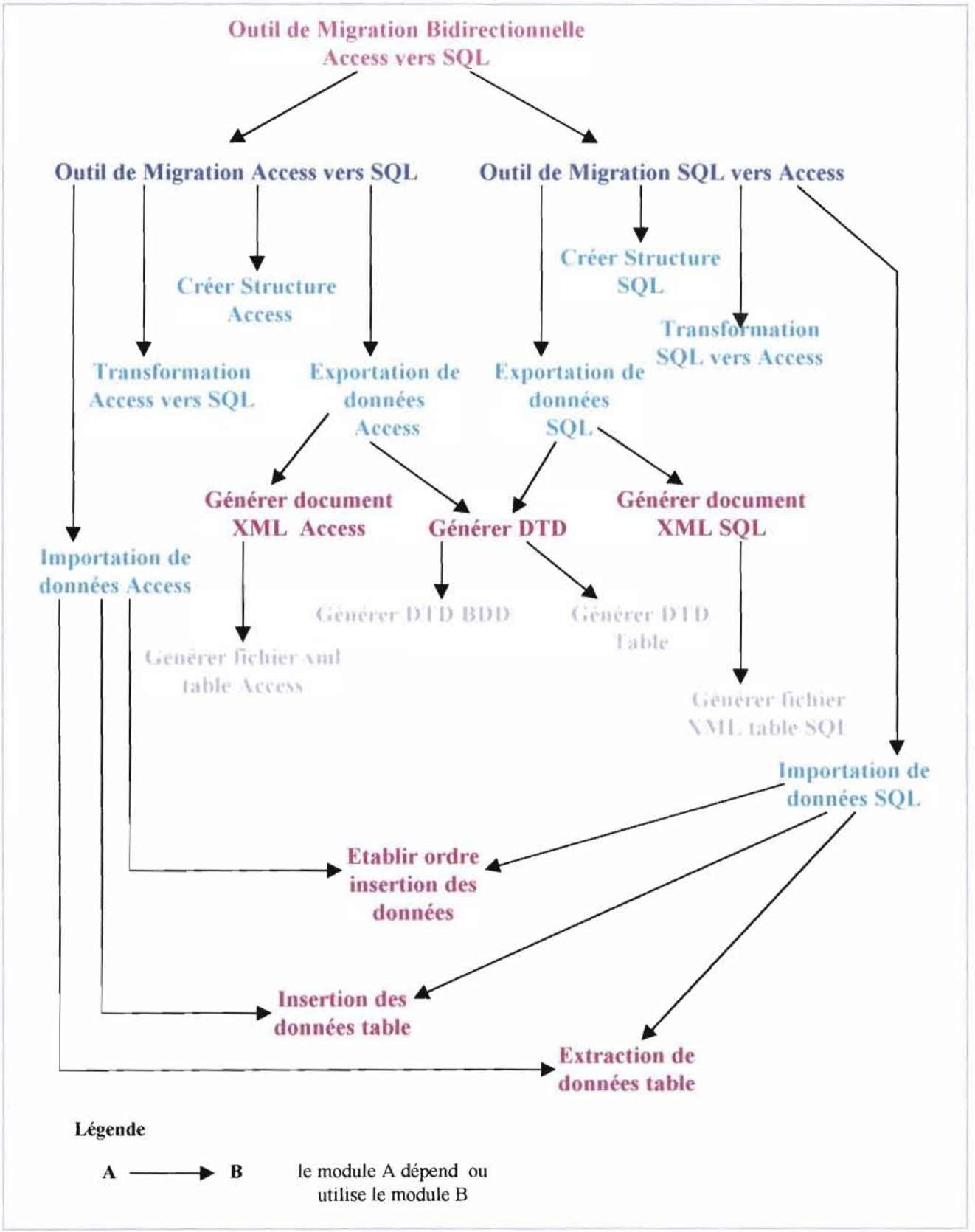


Figure 41 : dépendance des modules

IV) IMPLÉMENTATION

IV.1) Les différents choix de solutions

L'implémentation suppose le choix d'une solution réalisable. Ainsi nous opterons pour la solution n°3 au niveau de la spécification technique du problème car cette solution est plus flexible dans son implémentation. En effet la solution n°3 utilise l'approche XML qui permet de manipuler dans une seule entité (document XML) les structures de données et les données. Cette solution préconise essentiellement trois étapes qui sont :

- l'exportation de données sources;
  Elle consiste à représenter les données sources et leurs structures de données dans un fichier (document XML) qui respecte un formalisme de représentation de balises défini au préalable. Ce formalisme de représentation est traduit par une grammaire propre aux données.
- la transformation des données;
  Cette étape consiste à partir d'un document XML bien formé et valide représentant les données sources pour constituer un document XML qui sera conforme dans sa structuration aux structures de données de la base de données destination.
- l'importation des données
  A partir du document XML obtenu lors de l'étape de transformation, on procède à l'extraction des données utiles afin de pouvoir alimenter les tables destination en données par des séries d'insertions tout en respectant les contraintes d'intégrité établies au niveau de la base de données.

Aussi au niveau du choix architectural les cas1 et cas2 seront pris en considération dans notre implémentation. Les cas1 et cas2 stipulent qu'on peut avoir en entrée la base de données source (respectivement la base de données source et la base de données destination) pour en déduire une nouvelle base de données destination contenant les données résultant du processus de migration (respectivement la base de données destination considérée en entrée et qui va contenir les données migrées).

IV.2) Décomposition fonctionnelle et description des traitements

Préambule

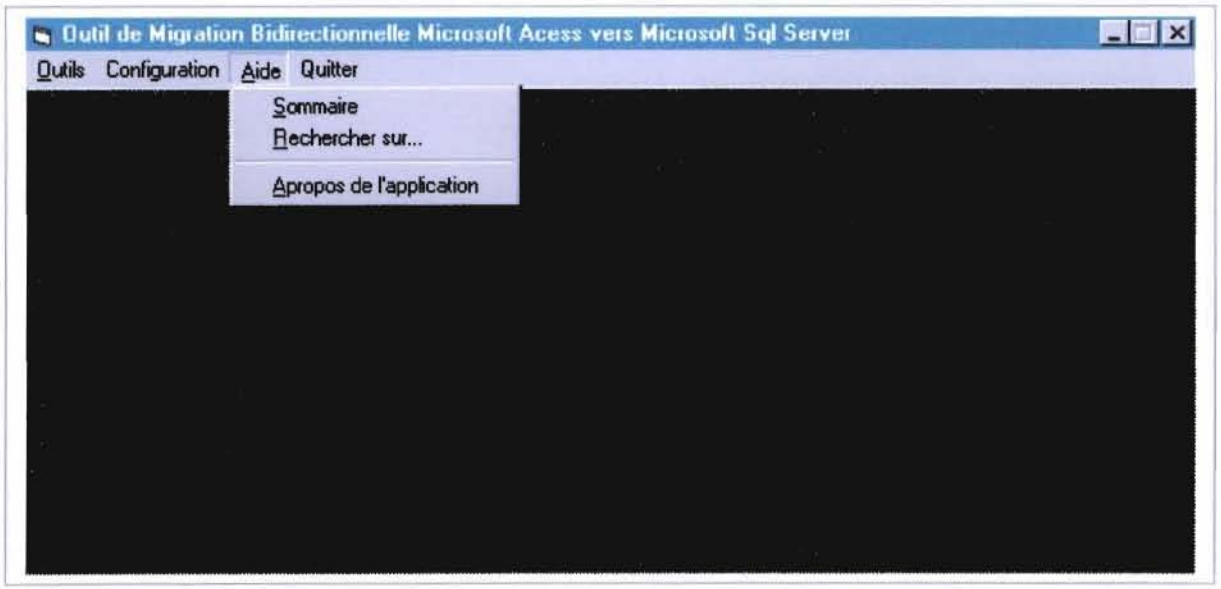
L'application à mettre au point comporte trois rubriques dont l'une — la rubrique de la transformation des données — est fonctionnelle. En effet cette rubrique de transformation des données est constituée de deux fonctions essentielles qui sont:

- *la migration de données Access vers SQL Server*
  Cette fonction doit permettre à partir d'une base de données sous Microsoft Access soit :
  - d'obtenir une nouvelle base de données équivalente sous l'environnement Microsoft SQL Server ;
  - de transférer les données de la base de données Access vers une base de données SQL existante .
- *la migration de données SQL Server vers Access*

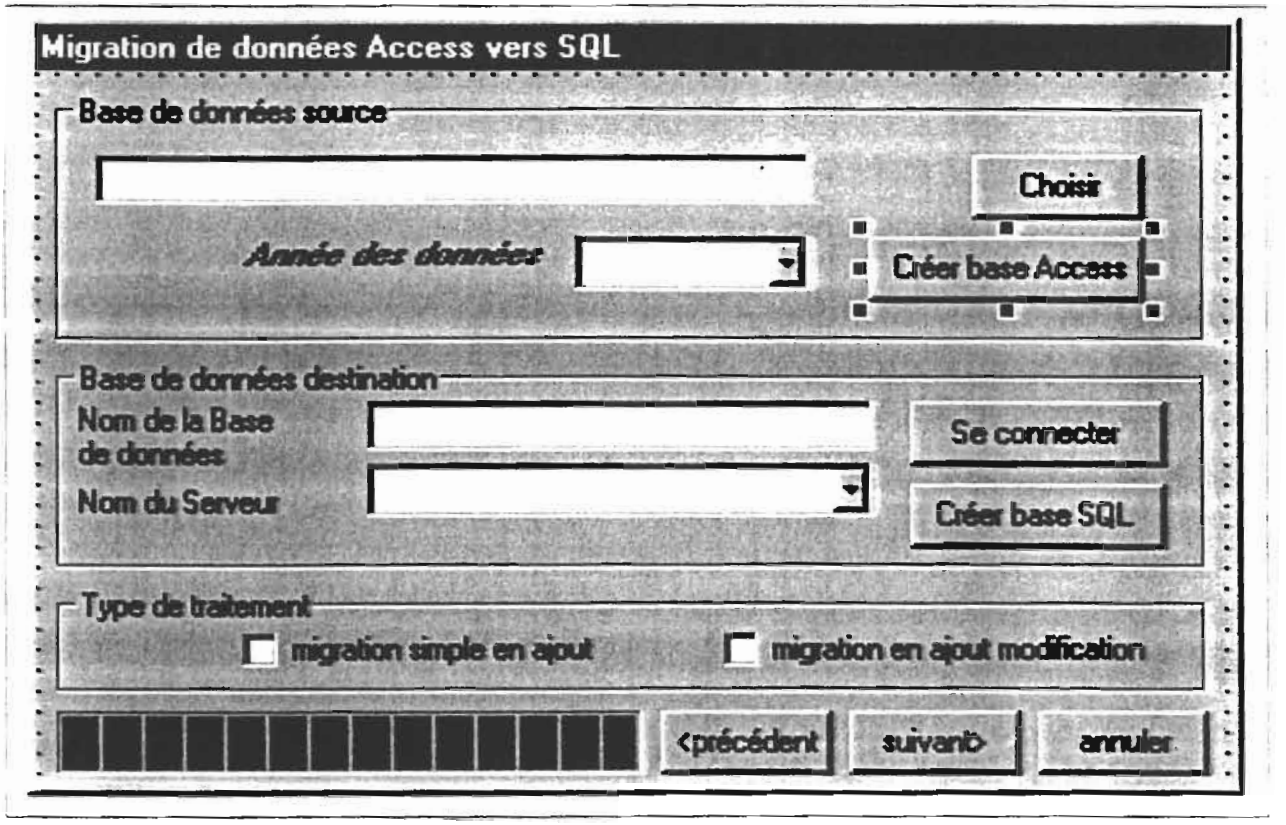
C'est une fonctionnalité inverse de la première car elle permet de transférer des données de l'environnement Microsoft SQL Server vers l'environnement Microsoft Access.

Dans cette section nous présentons le déroulement de l'exécution des fonctionnalités de l'application réalisées à travers les interfaces homme-machine et la description des différents traitements qui sont associés.

L'écran d'accueil de l'application



Le menu général



1- Créer structure SQL

Cette classe permet, à partir de la structure Access, de créer une structure multi-annuelle sous l'environnement SQL Server 7. Elle permet la sélection des tables qui seront des tables multi-annuelles dans la base SQL par l'introduction d'un champ année dans la clé primaire.

Contrairement aux outils natifs de migration de données qui suppriment toutes les contraintes définies sur le schéma, l'outil de migration mis au point ici conserve le maximum de contraintes proprement définies au niveau du schéma de la base de données.

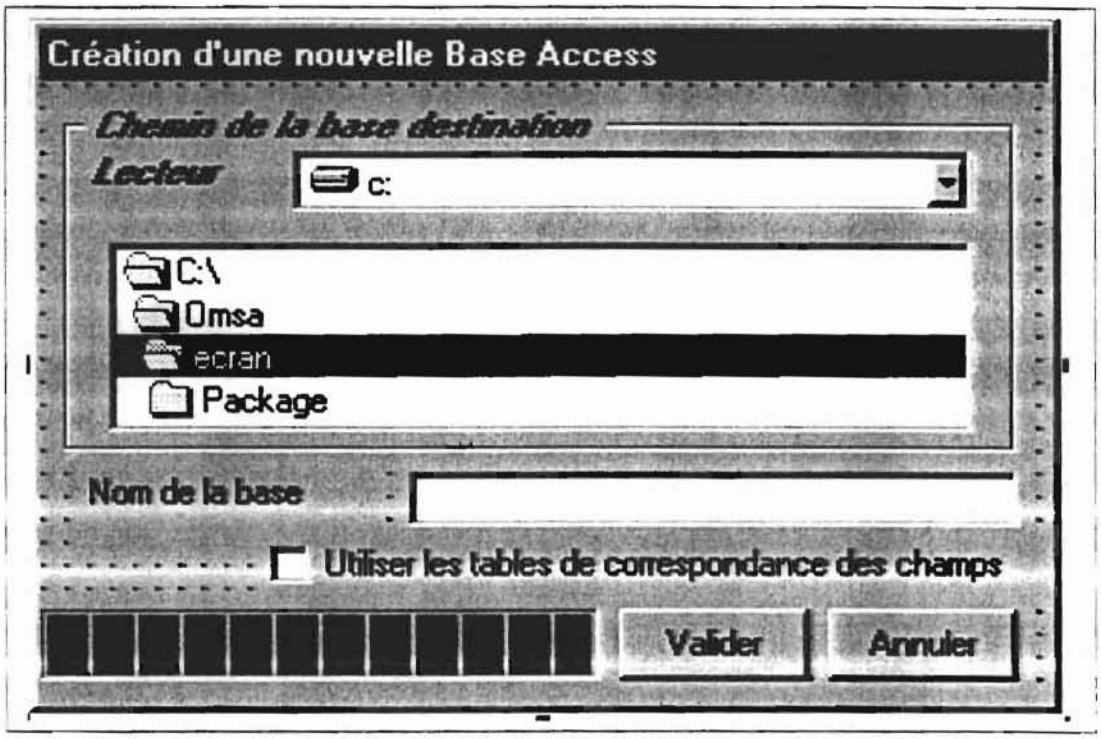
Créer Structure SQL
Lire Structure Access (BDD Access) *Lire les contraintes de la BDD Access (BDD Access)* *Générer Script SQL (BDD Access)* *Créer une Base SQL(Fichier_Script)* *Créer table SQL (X:table Access,Y: table SQL)* *Etablir Relation entre Tables(X, Y: table SQL)*

Description des traitements associés

Début Créer Structure SQL

1. *Identifer les tables isolées*
2. *Générer le script SQL correspondant au schéma de la base de données Access*
3. *Créer la base SQL en exécutant le script*

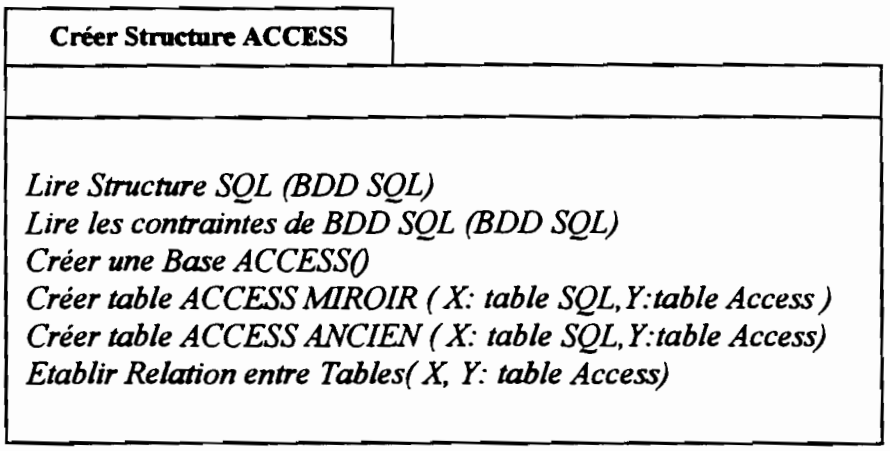
Fin Créer Structue SQL



2- Créer Structure Access

Cette classe permet, à partir de la structure multiannuelle SQL, de créer une structure annuelle dans un format Microsoft Access dans le but de pourvoir contenir les données pour une année. Cette classe lit le schéma de la base de données multi-annuelle SQL Server ainsi que les contraintes et en déduit une structure Access non multi-annuelle. L'outil de migration mis au point doit conserver le maximum de contraintes proprement définies au niveau du schéma de la base de données SQL.

Il existe deux variantes de cette classe : l'une permettant la création d'une structure miroir tandis que l'autre essaie de travailler dans un environnement particulier en ce sens qu'elle utilise une base de données statique qui contient le mapping des champs qui ont été modifiés. Cette fonctionnalité a été introduite dans le souci de pouvoir maintenir les frontaux existants sous Microsft Access.



Description des traitements associés

Début Créer Structure Access

1- Etablir une connexion à la base de données SQL
2- Pour chaque table appartenant { table de la base données SQL} faire
Créer tables access correspondantes ;
Fin pour
3- Pour chaque contrainte appartenant { contrainte de la base de données SQL } faire
Insérer la contrainte dans le schéma de la nouvelle base ;
Fin pour
4- fermer la connexion

Fin Créer Structure SQL

3- Exportation des données Access

Cette classe permet de constituer un document XML qui mémorise la structure et les données de la base Access. Le fichier obtenu servira après l'opération de transformation de source de données textuelles pour alimenter la base SQL Server.

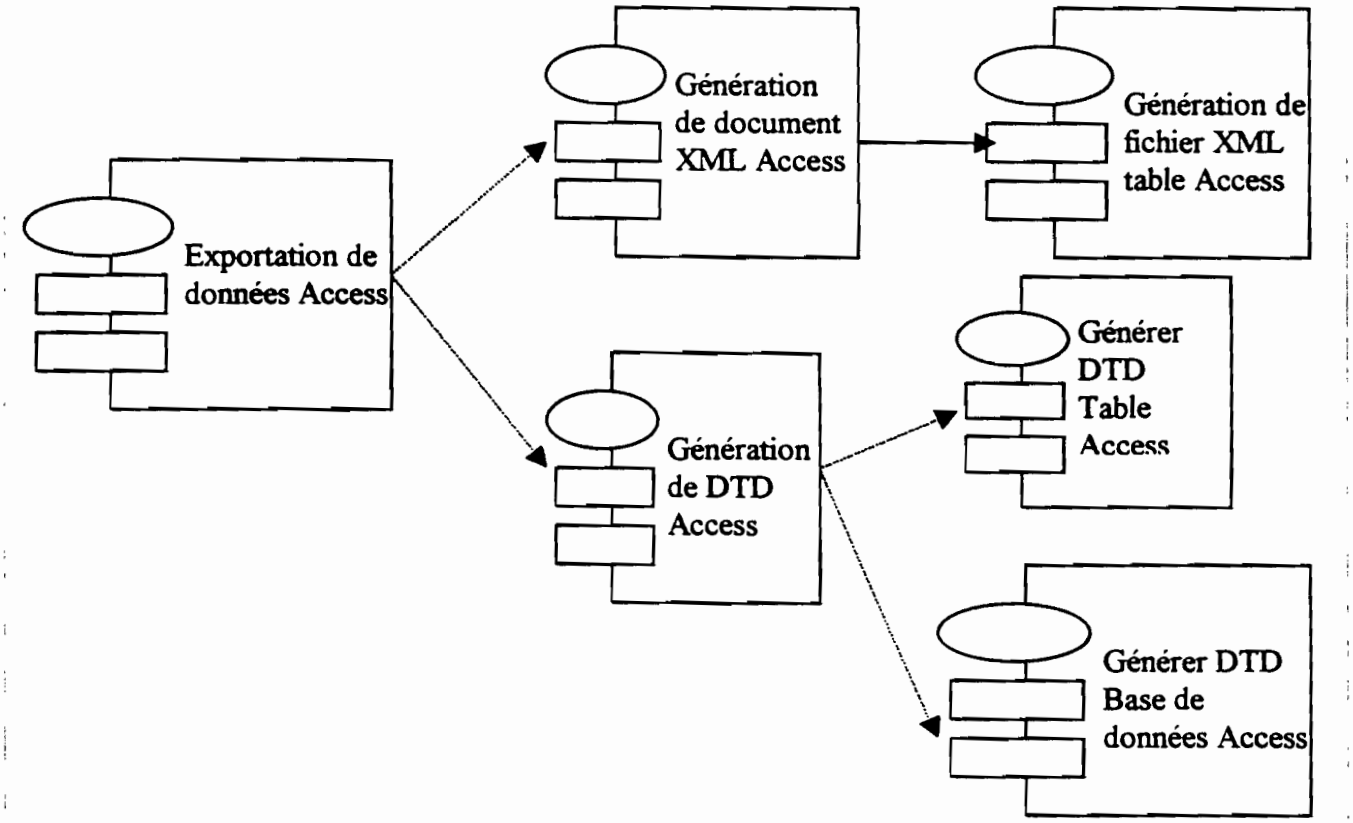


Figure n°42 : représentation de la décomposition du composant exportation de données Access

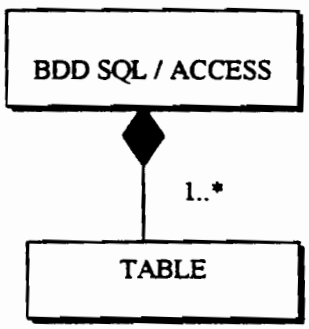


Figure n°43 : Schéma global de la base de données

Structure des différents DTD (Document Type Definition)

DTD BDD ACCESS/SQL

< ! ELEMENT BDD SQL /ACCESS (TABLE 1|TABLE2|.....| TABLEi)* >
<!ENTITY % E_TABLE1 SYSTEM " ***reference externe*** " >
.....
<!ENTITY % E_TABLEi SYSTEM " ***reference externe*** ">
< ! ELEMENT TABLE1 % TABLE 1 >
< ! ELEMENT TABLE 2 % TABLE 2 >
.....
< ! ELEMENT TABLEi % TABLE i >

DTD TABLEj

< ! ELEMENT TABLE j (Champ 1, champ 2 ,....., champ m) >
< ! ELEMENT Champ 1 (#PCDATA) >
< ! ELEMENT Champ 2 (#PCDATA) >
.....
< ! ELEMENT Champ m (#PCDATA) >

Description des traitements associés

Début Exportation Access

Sélection de la base de données source
Vérifier l'existence de la base de données sélectionnée
Générer la DTD de la BDD Access
Générer le document XML correspondant aux données de la base

Fin Exportation

Raffinement Générer DTD de la BDD Access

Ouvrir la base de données
Pour chaque table dans la base de données faire
Créer la DTD de la Table;
Fin faire
Créer la DTD de la Base de données
Fermer la base de données

Fin Raffinement Générer DTD de la BDD Access

Raffinement Générer le document XML Access

Ouvrir la base de données
Ouvrir le fichier résultat
Initialiser le fichier résultat
Pour chaque table sélectionnée dans la base de données faire
Créer le fichier de données associé à la table
Ajouter les données au fichier résultat
Fin faire
Fermer le fichier résultat
Fermer la base de données

Fin Raffinement Générer le document XML Access

Intégration du moteur xml

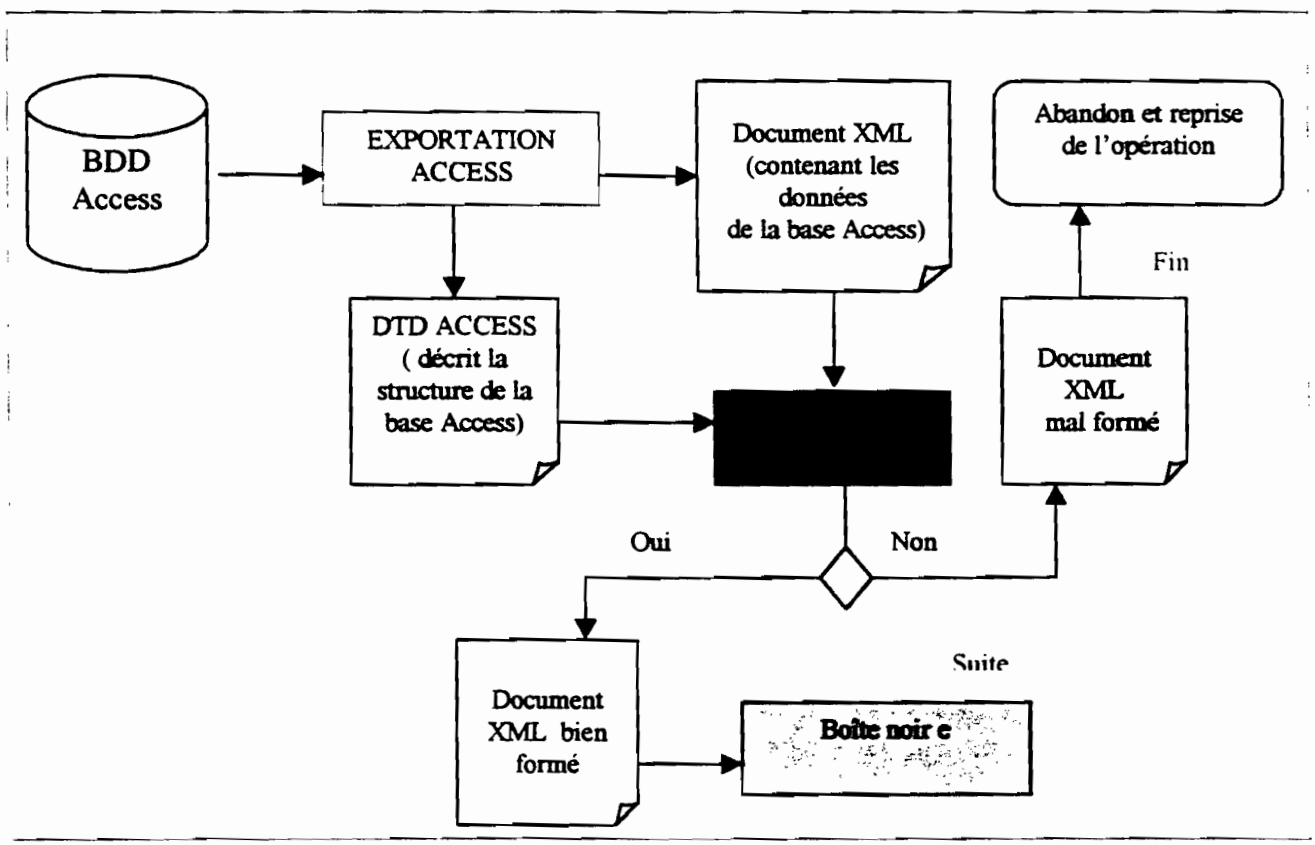


Figure 44 : processus de validation d'un document XML

Processeur XML

Un module logiciel appelé **processeur XML** est utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. On suppose qu'un processeur XML effectue son travail pour le compte d'un autre module, appelé **l'application**. Cette spécification décrit le comportement requis d'un processeur XML, c'est à dire la manière dont il doit lire des données XML et les informations qu'il doit fournir à l'application.

4-Module de la transformation (access vers SQL / SQL vers access)

L'incapacité du formalisme XML à intégrer les types de données nous a conduit à constituer dynamiquement une structure de données représentant les dictionnaires de données des bases de données source et destination afin de pouvoir définir de façon plus aisée le mapping des colonnes des différentes tables de données à migrer. On peut schématiser le dictionnaire de données de la façon suivante :

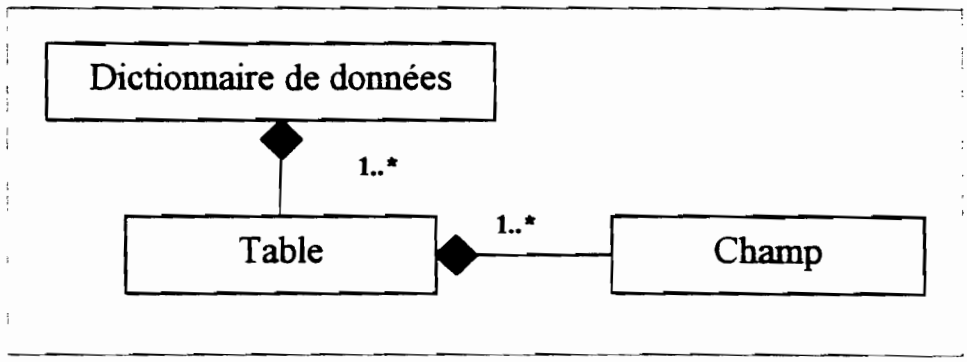


Figure 45 : Schéma du dictionnaire de données

Avant l'opération de la transformation il faut bien effectuer une opération qui consiste au chargement des variables c'est-à-dire premièrement à la constitution dynamique des différents dictionnaires de données des bases source et destination et à la définition de la correspondance des tables sources et destinations ainsi que le mapping des différentes colonnes.
Le résultat de cette opération est stocké dans une base de données qui constitue l'espace de travail de l'application. La structure de la base de données « espace de travail » est représentée de la manière suivante :

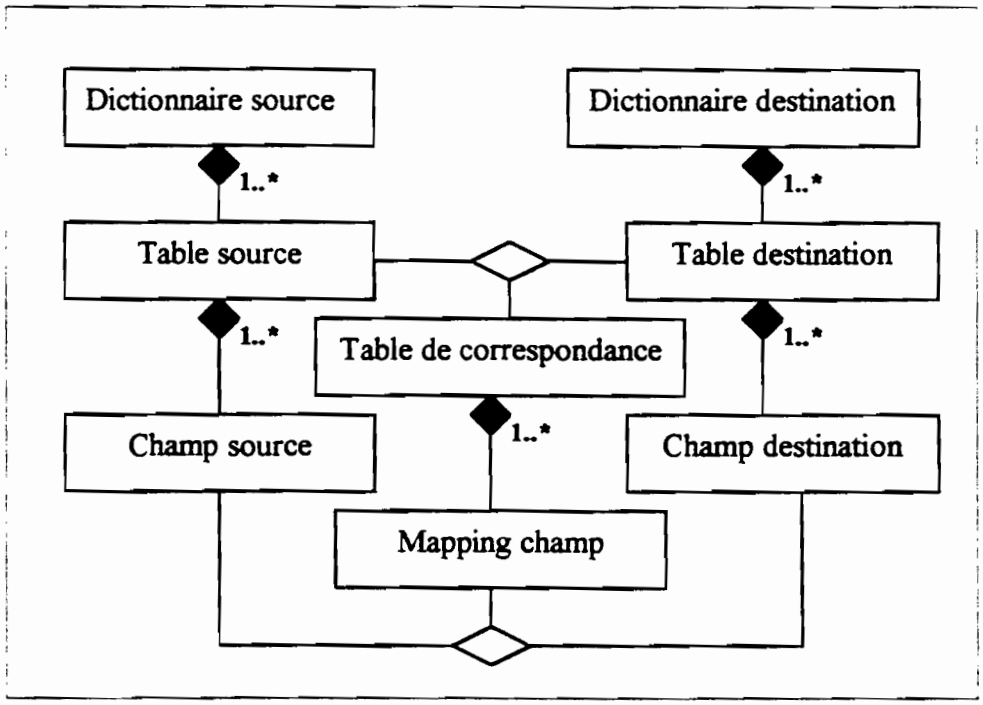
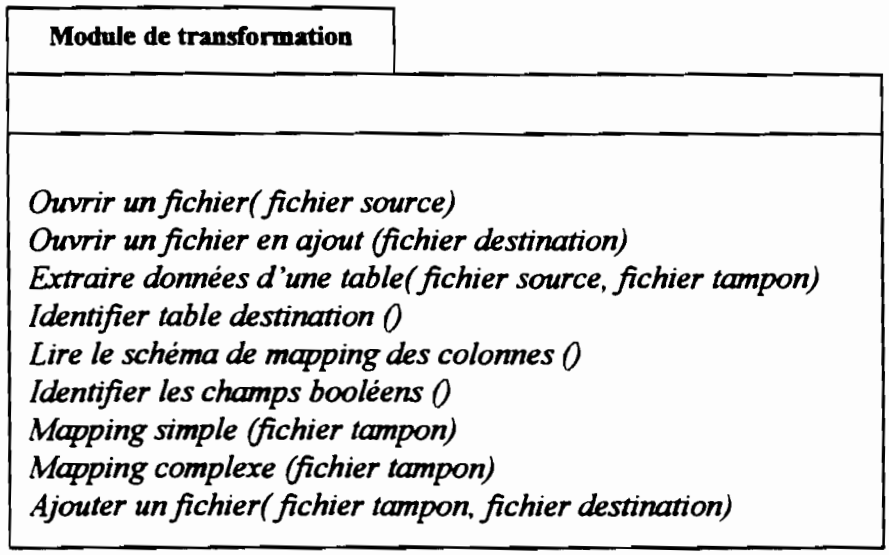


Figure 46 : représentation de l'espace de travail



Description des traitements associés

Transformation Access vers SQL server

1. *ouvrir la BDD source*
2. *liste := liste des tables ayant champ de type booléen*
3. *fermer la BDD source*
4. *ouvrir le fichier_export_access*
5. *ouvrir la table des correspondances*
6. *initialiser le fichier destination SQL*
7. ***récupérer la première entrée utile***
8. *Tant que not EOF(fichier_export_access) faire*

   Récupérer entrée table
   Vider (fichier tampon)
   Trouver table destination
   Extraction_traitement_données correspondantes à la table
   Ajouter fichier (fichier tampon , fichier destination SQL)

   Fin tant que

5 – Exportation des données SQL

Cette classe est simulaire à la classe *Exportation des données Access* en terme de fonctionnalités sauf que dans les traitements il y a souvent une grande différence du fait des formats des données qui ne sont pas les mêmes. De plus la structure de la base SQL est multi-annuelle.

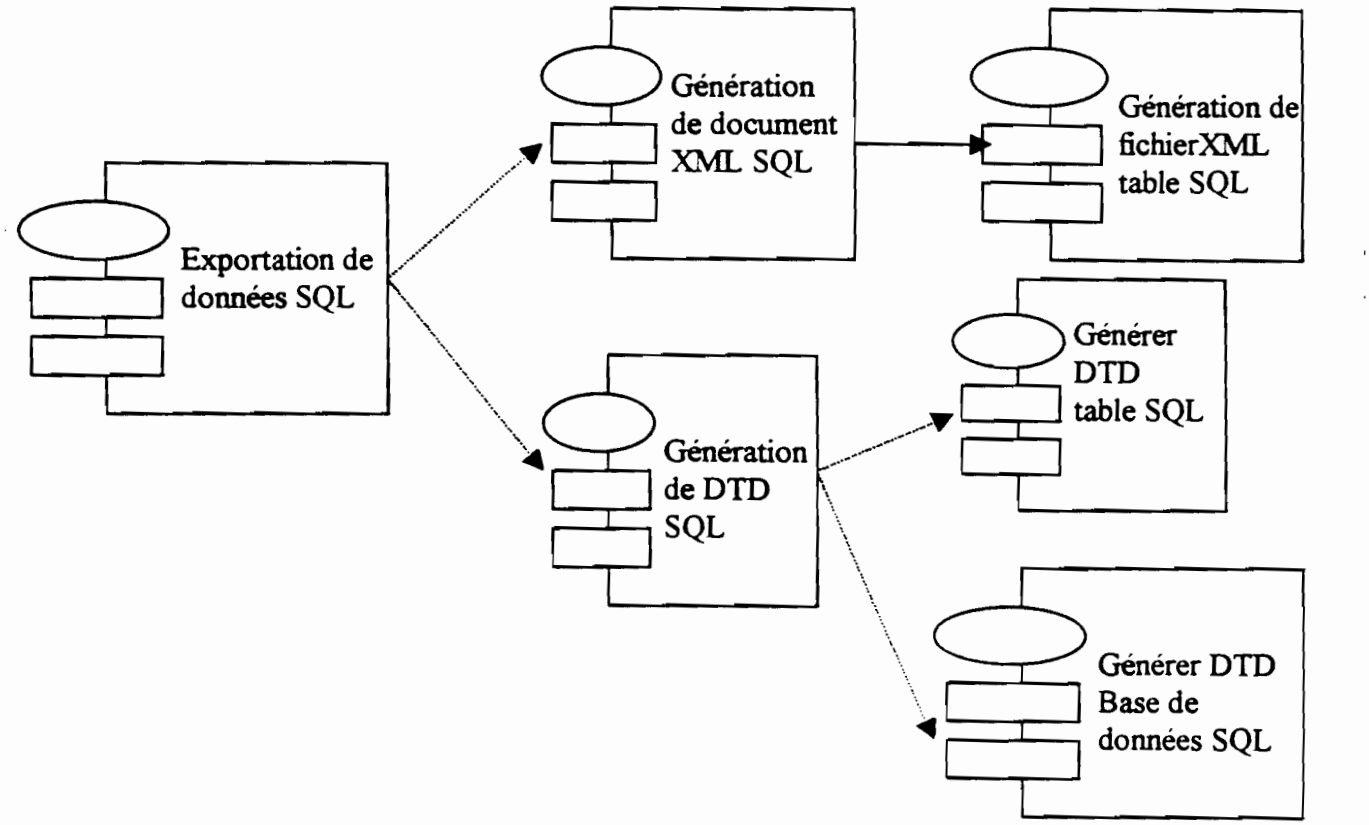
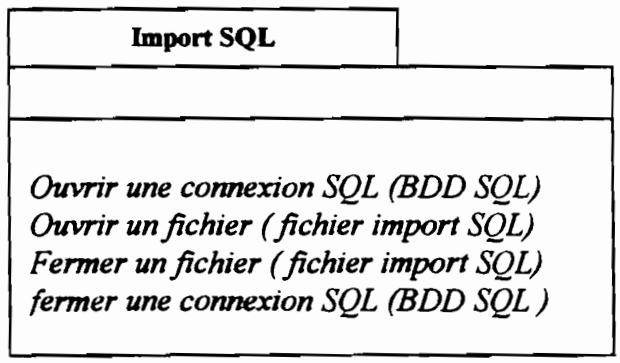


Figure 47 : représentation de la décomposition du composant exportation de données SQL

6 - import de données SQL

Cette classe permet à partir du fichier issu du module de transformation d'alimenter la base SQL Server en données. Avant toute opération d'insertion de données nous procédons par l'intermédiaire d'une autre classe à l'établissement d'un ordre d'insertion afin d'éviter des erreurs provenant des violations des contraintes d'intégrité référentielles.A partir du fichier représentant les données de la base SQL Server à travers une autre classe on extrait les données textuelles concernant chaque table qui seront stockées dans un fichier tampon. Ce fichier sera à son tour pris en entrée par un sous module qui se chargera de parcourir le fichier tout en extrayant les données de la représentation textuelle (données + structures de données), pour finalement procéder à des insertions dans la table correspondante.



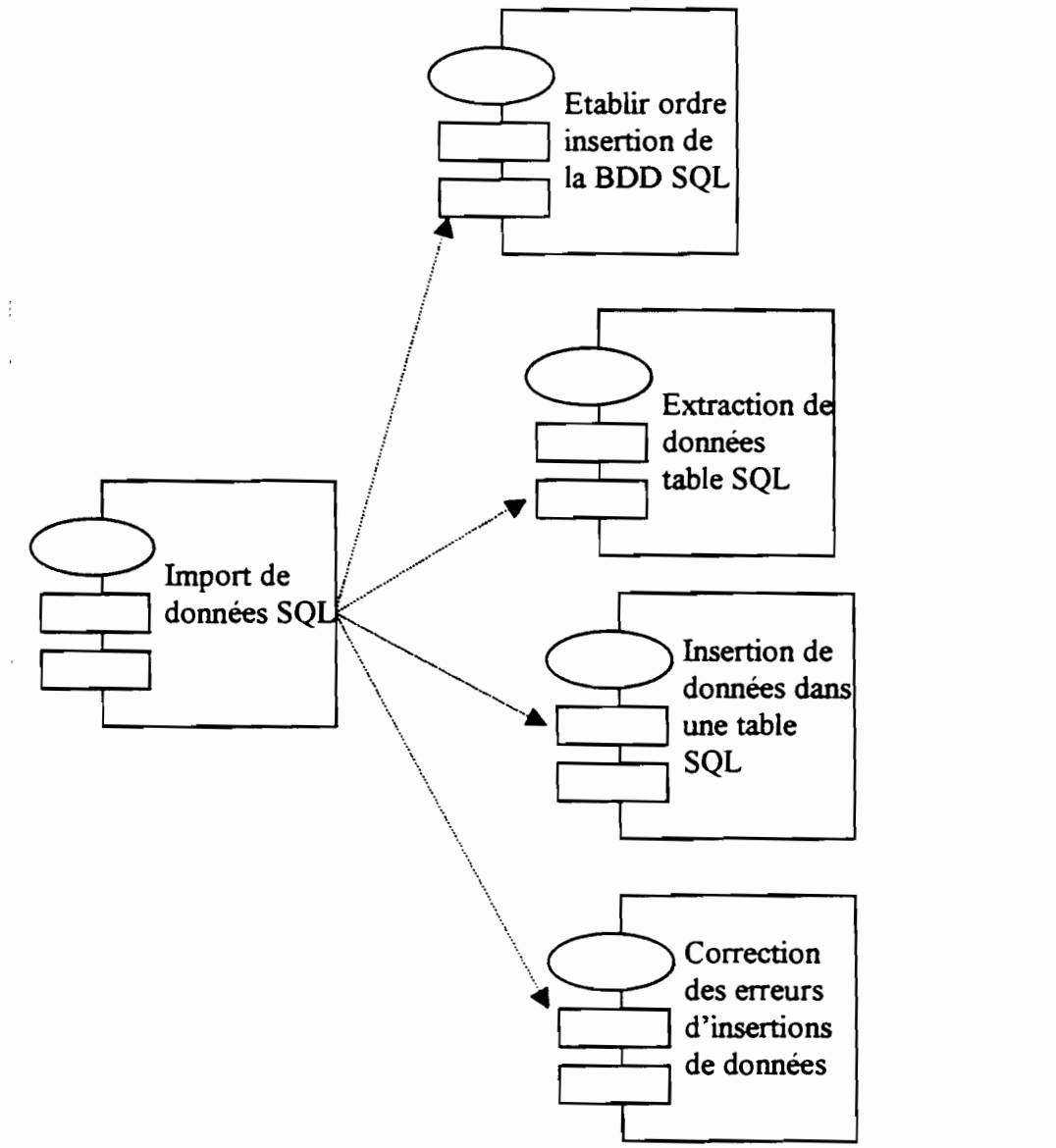


Figure 48 : décomposition du composant import données SQL

- Etablir ordre insertion
  Cette fonction s'avère très importante car dans la mesure où les contraintes d'intégrité référentielles établies entre les tables de la base de données destination doivent être conservées, il est nécessaire de pouvoir établir dynamiquement un ordre de la base de données de sorte que l'insertion des données dans les différentes relations ne soit pas source d'erreurs pour cause de violations des contraintes d'intégrité.
  Contrairement aux outils standards de migration existant sur le marché qui ne font aucun contrôle sur les données migrées comme en témoigne la suppresion de toutes les contraintes d'intégrité référentielles , de tous les index, de tous les déclencheurs et de toutes les clés.

Notre approche a pour avantage de conserver toutes les contraintes établies au niveau de la définition du schéma de données afin de pouvoir assurer le bon fonctionnement des différentes applications.
Dans cette fonction on utilise une stratégie de parcours en largeur d'abord. Ainsi on part des tables les moins contraignantes (celles qui n'ont pas de clés référentielles) pour finir par les tables les plus contraignantes.
Une racine virtuelle de l'arbre est une table fictive. Viennent ensuite dans l'arborescence les tables qui n'ont pas de clés référentielles. On réitère l'opération et on n'insère une table que lorsque toutes tables auxquelles elle fait référence ont été insérées.

Description de l'algorithme de « Etablir ordre insertion »

Début Etablir ordre insertion

1. identifier les tables qui n'ont pas de clés étrangères $T_1, T_2, .., T_k$
2. Insérer ces tables à partir du nœud racine virtuelle
3. Soit ***Liste non traitée*** Constituée de la liste des tables de base de données privées des tables n'ayant pas de clés étrangères
4. Répéter
   a. Pour chaque table appartenant à ***Liste non traitée*** faire
      Etablir la liste des tables référencées par la table en cours de traitement (soit *Liste référencée table* cette liste) ;
      Fin pour
   b. Si ***Liste référencée table*** $\subseteq$ Liste des tables constituant l'arbre ainsi créé alors
      Ajouter la table dans l'arbre ;
      Retirer la table de la ***Liste non traitée*** ;
      Finsi
   c. Si Nombre de passe >10 alors
      Sortir de la recursivité
      Finsi

Jusqu'à ce que la ***Liste non traitée*** = liste vide

Fin Etablir ordre insertion

- Extraire les données table
- Insertion données dans une table SQL

7 - import de données access

Cette classe a les mêmes fonctionnalités que la classe *Import de données SQL* mais la seule différence réside dans les traitements car la philosophie de résolution est la même .

8 - Modules des erreurs

L'opération de migration est une opération très délicate au cours de laquelle il peut y avoir des erreurs. Il y a lieu d'implémenter des stratégies de reprise d'erreurs éventuelles lors d'une opération de transfert de données. Pour cela il faut situer les sources possibles d'erreurs. Des erreurs peuvent survenir dans les cas suivants :

- la violation des contraintes d'intégrité référentielles ;
- l'incohérence dans la définition de la correspondance des champs source et destination (configuration de l'espace de travail ou workspace) ;
- plusieurs champs sources dans la même table ont le même champ destination ;
- incohérence de types de données ;

La stratégie générale est la reprise totale de l'opération, mais on peut aussi, à partir du fichier d'erreurs identifier les causes possibles des erreurs et ensuite procéder à la sélection des tables manquantes ou à la redéfinition de la configuration de l'espace de travail de l'application de façon à éviter les sources d'erreurs evoquées ci-dessus .

IV.2) Les principes d'optimisation

Définition des performances d'une application

Le terme «performances», lorsqu'il concerne une application, recouvre plusieurs aspects, parmi lesquels:

- les performances réelles;
- les performances apparentes;
- les performances constantes et fiables.

Les performances réelles ont trait à la vitesse effective à laquelle l'application calcule ou effectue des opérations. Pour aboutir à une application très rapide, il est possible combiner différentes techniques d'optimisation du code.

Les performances apparentes sont celles perçues par l'utilisateur quant à la vitesse de réaction de l'application. Bien que le plus souvent liées aux performances réelles, elles en diffèrent par le fait qu'elles peuvent recourir à des techniques opposées. Par exemple, l'application peut utiliser un curseur de grande taille côté client pour éviter de fréquents accès aux bases de données et un trafic réseau important. Le curseur côté client offre un temps de réponse utilisateur extrêmement rapide, avec l'inconvénient d'un trafic sur le réseau plus important et d'un temps de réponse initial plus lent.

Un curseur est une structure de données permettant de parcourir les enregistrements d'une table.

Les performances constantes et fiables caractérisent une application stable, évolutive et toujours disponible. Cela implique certains choix en matière de conception qui peuvent affecter les performances réelles. Par exemple, l'utilisation de MSMQ (Microsoft Message Queue Server) pour la gestion de la mise en file d'attente entraîne une légère baisse des performances due à cette

surcharge de traitement. L'avantage, en revanche, est qu'au fur et à mesure de l'augmentation de la charge, l'application évolue en douceur et continue à effectuer son travail.

Les applications les plus performantes sont conçues pour satisfaire tous ces objectifs de performances à la fois. Bien entendu, cela n'est pas toujours réalisable, mais un accès aux données très performant requiert que la conception de l'application tienne compte de tous ces objectifs le plus souvent possible.

Limitation des accès disques

Même un disque rapide ralentira l'application si les Entrées/Sorties (E/S) disque sont trop nombreuses. Si le temps d'utilisation du disque et la longueur de la file d'attente du disque augmentent du fait d'une activité excessive en lecture-écriture, l'accès disque peut constituer un problème de performances.
Si l'on examine l'accès disque en relation avec les performances, il est nécessaire de considérer de nombreux facteurs complexes du sous-système d'Entrées/Sorties disque. Il faut prendre en compte le type et le nombre des contrôleurs de disque, le type des lecteurs de disque et les options nécessaires pour les configurations qui tolèrent des pannes. Les performances générales du système peuvent être affectées de façon spectaculaire par ces choix de configuration.

Pour augmenter les performances de l'accès disque, il est important d'utiliser tous les canaux SCSI disponibles et d'en ajouter d'autres, le cas échéant. L'ajout de lecteurs de disque supplémentaires sera toujours bénéfique pour les performances, particulièrement si les E/S disque sont par nature aléatoires. Tous les lecteurs ont des limitations mécaniques qui affectent leurs performances lors des sauts de piste en piste ou de secteur en secteur. En ajoutant des lecteurs supplémentaires, l'accès disque peut être géré de façon plus efficace car certaines E/S physiques sur le disque peuvent intervenir en parallèle.

Il est possible d'analyser les performances des E/S disque à l'aide d'un analyseur de performances. S'il indique des demandes d'E/S excessives sur le disque (ou une tendance à la hausse), il faut évaluer l'application pour déterminer quand et pourquoi surviennent les accès disque. Il est parfois possible de modifier la stratégie globale d'accès aux données afin de réduire de façon significative l'utilisation du disque.
Au cas où le disque constitue un goulet d'étranglement pour les performances, on peut résoudre ce facteur limitant par :

- la restructuration de la quantité ou du type des E/S disque;
- l'ajout de RAM pour augmenter la taille des mémoires tampon d'E/S de la base de données ou des fichiers;
- l'utilisation de plusieurs contrôleurs de disques ;
- l'ajout d'un ou plusieurs contrôleurs de disques en mémoire cache pour y mettre les opérations de lecture et d'écriture;
- la limitation ou le redéploiement de certains services d'application.

Choix des meilleurs composants d'accès aux données

La question que se pose le développeur est souvent la suivante: «quelle technologie d'accès aux données dois-je utiliser pour créer cette application d'entreprise?». Pour pouvoir répondre à cette question, il faut considérer deux aspects: l'importance de la réutilisation du code et la capacité du développeur à mettre en œuvre l'interface choisie. Il arrive souvent que le développeur, en quête de meilleures performances, mette en œuvre une solution d'accès aux données alambiquée aboutissant seulement à un travail de maintenance accru de l'application. Les technologies d'accès aux données les plus modernes réduisent généralement le temps de développement et simplifient le code, tout en offrant de bonnes performances et en répondant aux besoins exprimés en matière de fonctionnalité.

On peut utiliser de façon efficace pratiquement toutes les technologies d'accès aux données dans la plupart des situations.

Réduction du code

Si la réduction de la taille d'une application est un facteur important, il est possible d'appliquer plusieurs techniques pour rendre le code plus compact. Outre le fait de réduire la taille de l'application en mémoire, la plupart de ces techniques réduisent également la taille du fichier exécutable. Autre avantage : le chargement d'une application de taille plus petite est plus rapide.

La plupart des techniques d'optimisation impliquent la suppression des éléments non nécessaires de notre code.

En général, il convient de rationaliser des éléments, comme les variables, les feuilles et les procédures, qui occupent beaucoup d'espace mémoire. Plusieurs techniques permettent de réduire la mémoire occupée par une application lorsqu'elle est exécutée sous forme de fichier .exécutable. Les techniques suivantes permettent de réduire la taille du code :

- *La Réduction le nombre de feuilles chargées ;*

Chaque feuille chargée, visible ou non, consomme une quantité de mémoire importante (qui varie avec le nombre et les types de contrôles de la feuille, la taille des images bitmap sur la feuille, etc.).

Principe : charger uniquement les feuilles qui doivent être affichées ; et les décharger lorsqu'elles deviennent inutiles (au lieu de les masquer).

- *L'organisation des modules ;*

Les outils de développement charge les modules sur demande. Autrement dit, un module n'est chargé en mémoire que si le code appelle une des procédures de ce module.

Principe : placer les procédures associées dans le même module.

- *L'utilisation de tableaux dynamiques et la récupération de la mémoire par des opérations d'effacement ;*

L'utilisation des tableaux dynamiques présente plus d'avantages que celle des tableaux fixes.

Principe : supprimer les données inutiles d'un tableau dynamique quand elles ne sont plus utilisées afin de récupérer l'espace mémoire occupé.

- *La Récupération de l'espace occupé par des chaînes ou des variables d'objets ;*

L'espace utilisé par des variables de tableaux et de chaînes locales (non statiques) est récupéré automatiquement une fois la procédure terminée. Cependant, les variables de chaînes

globales au niveau du module ainsi que les variables tableaux sont conservées pendant toute la durée d'exécution du programme. Si on désire conserver une application aussi petite que possible, on doit récupérer l'espace utilisé par ces variables dès qu'on le peut.

- *Éliminer le code désactivé et les variables inutilisées.*

Pendant le développement et la modification d'une application, on peut avoir laissé un *code désactivé*, c'est-à-dire des procédures entières qui n'ont pas été appelées dans notre code. Il est possible aussi d'avoir déclaré des variables inutiles. La plupart des outils de développement ne supprime ni les variables inutilisées ni le code désactivé lorsqu'on crée un fichier exécutable. Pour cela on doit revoir le code pour rechercher et supprimer les procédures et variables inutilisées.

Stratégie d'optimisation utilisée

➢ *Choix d'une technologie d'accès aux données orientée performance*

Si l'application d'entreprise doit probablement accéder à des données distribuées dans toute notre organisation, y compris sur des ordinateurs mainframe, sur des stations de travail locales et dans des fichiers distribués distants, il peut arriver à traiter des données hétérogènes. Ces magasins de données peuvent utiliser un format différent, un moteur de base de données différent et être accessibles d'une façon particulière. L'utilisation efficace de différents magasins de données est un problème complexe de programmation d'application qui implique des choix en matière de conception architecturale et de technologie d'accès aux données, ainsi que des considérations relatives aux performances.

Bien qu'il existe de nombreuses et excellentes stratégies de mise en œuvre pour un accès très performant aux données, pour la résolution du problème qui nous a été soumis, nous avons opté pour les composants ActiveX® Data Objects (ADO) comme technologie d'accès aux données. OLE DB avec ADO propose une solution uniforme d'accès aux données pour toute une gamme de structures de fichiers et de magasins de données. Cette combinaison offre toujours un accès aux données rapide, évolutif et facile à gérer car les paramètres de connexion aux sources de données sont transparents.

➢ *La réduction du nombre de contrôles*

Lorsqu'on crée une application, la limitation du nombre des contrôles qu'on place sur une feuille doit être une attitude à avoir à l'esprit. La limite réelle dépend du type des contrôles et du système disponible mais, en pratique, toute feuille comprenant un grand nombre de contrôles s'exécutera lentement. Il est préférable d'utiliser des groupes de contrôles chaque fois que cela est possible, au lieu de placer un grand nombre de contrôles de même type sur une feuille au moment de la création.

➢ *La limitation des accès disque*

Vu que la solution qu'on doit implémenter utilise au niveau des traitements des fichiers très volumieux il est opportun d'avoir une politique de limitation des accès disque. C'est ainsi qu'on procède à certains fragmentations de fichiers très volumieux avec des traitements particuliers pour éviter une double parcours des fichiers pour dégager un gain en terme de temps de traitement.

IV.3) Critiques de la méthodologie utilisée

- Avantages

L'approche XML permet une grande souplesse dans la manipulation de données et de leurs structures en une seule entité et cela favorise une représentation textuelle des données à migrer. Ainsi la conversion des données en format textuel est adaptée en terme de traitement à la plupart des langages de programmation. C'est ainsi qu'avec les méthodes classiques de traitement des fichiers on arrive à résoudre la problématique qui nous a été soumise.

- Inconvénients

L'approche de résolution du problème utilisé n'a pas que des avantages, elle possède aussi quelques insuffisances. C'est ainsi qu'il y a lieu de déplorer l'espace disque occupé par l'espace de travail de l'application. En effet les fichiers résultants de la représentation les données des bases de données sont relativement volumineux et cela peut souvent être un handicap majeur en cas d'insuffisance d'espace disque. Par exemple pour une base de données Access de 40 Mo de données, le fichier textuel codant cette base est de l'ordre de 100 Mo, soit un rapport de 2.5.

Mais il faut noter que l'espace d'exécution est aussitôt libéré après l'exécution de l'application.

CONCLUSION

➢ Bilan

L'outil développé a effectivement permis la migration de quatre années de données de Microsoft Access vers Microsoft SQL Server 7.

➢ Perspectives

Les données obtenues au niveau de la base de données SQL alimentent un frontal interrogeable pour obtenir en ligne les différents tableaux de bord stratégiques pour la Direction des Etudes et de la planification du MEBA.

Une maquette de site web, a été réalisée par un autre membre de l'équipe qui utilise les données issues de la migration.

Une étude considère actuellement l'hébergement sur serveur HTTP (HyperText Transfert protocol) d'un site ouvert.

➢ Apports

La réalisation de l'étude qui nous a été confiée au Projet d'Appui à l'Enseignement de Base était d'un intérêt particulier pour nous car elle a été d'un enseignement enrichissant.

Tout d'abord l'étude a permis d'utiliser une combinaison de méthodologies en l'occurrence l'approche XML, l'approche UML et la méthode Merise pouvant être considérée comme une démarche méthodologique propre à la résolution du problème spécifique qu'est la migration de données.

Les différents concepts utilisés, au niveau de l'abstraction, permettent de manipuler les structures de données et les données dans une seule entité facilitant ainsi notre approche de solution tout en utilisant les avantages de conceptualisation de l'approche orientée objet.

L'étude a également été l'occasion de mettre en œuvre les connaissances acquises au cours de la formation à travers la mise au point de l'application dénommée "outil de migration bidirectionnelle Microsoft Access vers Microsoft SQL Server", tout en répondant aux attentes de l'utilisateur.
Elle nous a enfin permis de travailler en équipe pour l'aboutissement du même objectif et cela est un pas essentiel pour notre insertion dans la vie professionnelle, où les relations interpersonnelles jouent un rôle fondamental dans la réussite d'un projet informatique.
Pour finir, au sortir de cette étude mon dernier sentiment est d'avoir beaucoup appris mais aussi qu'il reste beaucoup à apprendre. Ce sentiment fait de moi une personne qui se remet en cause chaque jour et me motive à envisager une poursuite éventuelle de mes études.

Références

bibliographiques

[1] ANDREW LAYMAN, JEAN PAOLI (Microsoft Coporation), STEVE DE ROSE (Inso Coporation), HENRYS THOMPSON (University of Edinburgh) : *Specification for XML-Data*, June 26, 1997

[2] C. DELOBEL , M. ADIBA : *Bases de données et systèmes relationnels*, DUNOD Informatique, juillet 1991

[3] C. GODART, F. CHAROY : *Bases de données pour le génie logiciel*, Masson décembre 1992

[4] D. MARTIN : *Techniques avancées pour les bases de données*, DUNOD informatique, 1985

[5] ELMASRI / NAVATTHE : *FUNDAMENTALS OF DATABAES SYSTEMS*

[6] GERARG FRANTZ, *Livre d'or Visual Basic 4.0 : Toutes les nouveautés (OLE automation, Module de classe, collection, contrôles OCX..)*, SYBEX, 18 septembre 1996

[7] GERARG FRANTZ, *Le Platinium : Visual Basic 6, Votre guide de Visaul Basic pour Windows 95, Windows 98 et Windows NT4, Applications MDI, Les API Windows, Le contrôleur OLE, les contrôles ActiveX, DDE, Le multitâche, Les DLL, Les contrôles de Windows*, SYBEX, décembre 1999

[8] GEORGES GARDARIN : *Les systèmes et leurs langages*, EYROLLES, 1991

[9] HATOM SMINE, *ORACLE : Architecture, Administration et Optimisation (oracle version 6)*, Eyrolles, décembre 1991

[10] HUBERT TARDIEU, ARNOLD ROCHFELS, GEORGES PANET, RENE COLLETTI : *La méthode MERISE Tome2 « Démarche et pratiques »*, LES EDITIONS D'ORGANISATION, 1991

[11] JOHN CONNELL : *Accès aux bases de données avec Visual Basic 6 : Programmation ADO 2.0*, Eyrolles, avril 200

[12] JEAN PATRICK MATHERON : *Comprendre MERISE « outils conceptuels et organisationnels »*, EYROLLES, 1992

[13] MICROSOFT : *Administration système pour Microsoft SQL Server 7.0 Manuel de travail du stagiaire*, Ref : n° x 04 – 15727, 03/1999

[14] MARC ISRAËL, *SQL Server 7 : Guide de l'administrateur et du développeur* Eyrolles, janvier 2000

[15] MARC HUMBERT : *Les bases de données*, HERMES, 1991

[16] NASSER KETTANI, DOMINIQUE MIGNET, PASCAL PARE, CAMILLE ROSENTHAL-SABROUX : *De Merise à UML*, EYROLLES, 1999

[17] REMY LENTZNER, *Microsoft Visual Basic 6 : les bases de données et SQL Server 7*, Eyrolles, 7 mars 2000

[18] W3C, *[W3C : Extensible Markup Language (XML) 1.0]* : W3C Recommendation, 10 -Feb 1998

Internet

1. http://www.w3.org/TR/Rec-XML/
   Ce site contient les documents de référence de l'approche XML, issus de la normalisation de l'OMG (Object Management Group).
2. http://www.w3.org/XML
   Ce site contient des informations plus larges de la technologie XML.
3. http://www.wdvl.com/software/XML
   ce site donne un aperçu de la technologie XML et des domaines applications.
4. http://msdn.microsoft.com
   Ce site contient les manuels de référence des outils des environnements de développement des produits Microsoft tel Visual Basic.
5. http://www.webdevelopper.com/XML
   Ce site montre des exemples d'applications réalisées à l'aide de l'approche XML.
6. http://wdvl.internet.com/software/XML
   Ce site montre les domaines d'application possibles de la technologies xml.
7. htpp://www.megginson.com/sax/sax1
   Ce site propose des informations sur l'approche XML, mais des versions de processeurs xml pour la validation des documents XML. Par exemple sax1
8. http://www.citeweb.net/apetitje/XML
   Ce site contient des exemples comentées de structuration des documents avec l'approche XML.
9. http://www.microsoft.com/vbasic
   Ce site contient les documents de références de Visual Basic, ainsi que des exemples sur le développement de certains contrôles.
10. http://msdn.microsoft.com/library
    Ce site contient des informations relatives à la librairie des différents outils de développement de Microsoft.

ANNEXES

Annexe I : Présentation de l'organisme

La coopération française au Burkina Faso

La France est l'un des principaux bailleurs de fonds au Burkina Faso. Sur la période allant de 1988 à 1997, la part française a représenté près d'un quart de l'Aide Publique au Développement (APD) reçue par le Burkina Faso soit environ 400 millions de Francs (FF).

En 1996, la France a décidé de consacrer une partie de son APD au développement de l'éducation de base. C'est ainsi que le Projet d'Appui à l'Education de Base (PAEB) a été créé. Celui-ci a pour missions de participer à l'analyse de la demande sociale, de soutenir la dynamisation du réseau d'animation pédagogique, de développer des outils de planification (carte scolaire, gestion du personnel) et d'apporter une aide pédagogique à l'enseignement non formel. Le projet implique 10 assistants techniques français (ATF) et est piloté par deux coordinateurs (un burkinabé et un français). Le projet est structuré en 4 composantes :

- Evaluation du système éducatif (C1),
- Rénovation pédagogique, dispositif de formation (C2),
- Renforcement du système d'information, appui à la décentralisation et au management des ressources humaines (C3),
- Appui dynamique de l'éducation non formelle (C4).

La composante C3, constituée d'un ATF et d'un Coopérant du Service National (CSN), est celle concernée par le projet du stage. Son rôle, défini en 1996, consistait à mettre en place un nouveau système de collecte et de production informatisé des statistiques scolaires annuelles. Le système d'information de l'Enseignement de Base étant aujourd'hui réalisé, les nouvelles tâches de la composante consistent à l'extension à d'autres systèmes éducatifs (secondaire, non formel), à l'élaboration de la carte scolaire ainsi qu'au suivi de la collecte et de la saisie des données. L'objectif de cette composante est de fournir aux décideurs de l'éducation (Etat, ONG, coopérations bilatérales) des informations sur la situation du système éducatif afin d'établir la planification des besoins et surtout des ressources.

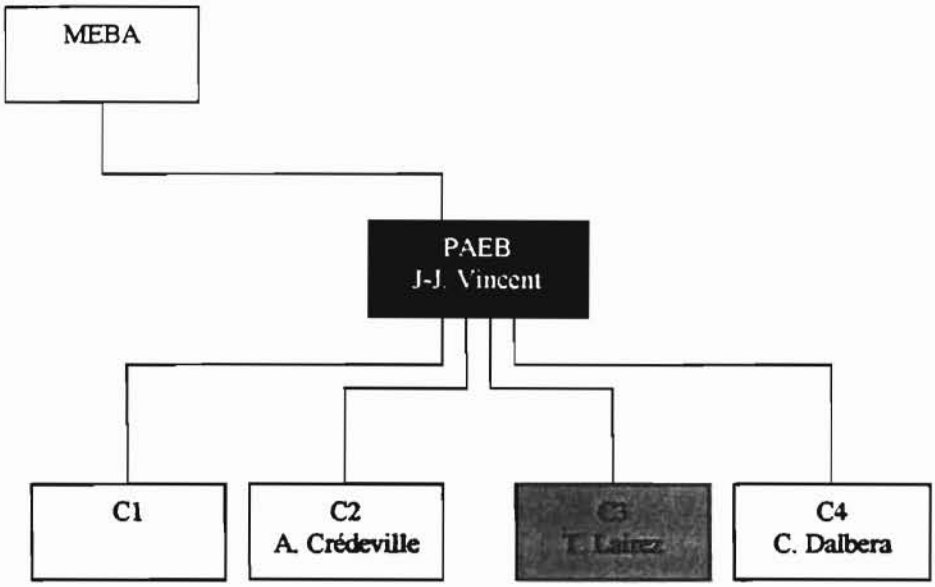
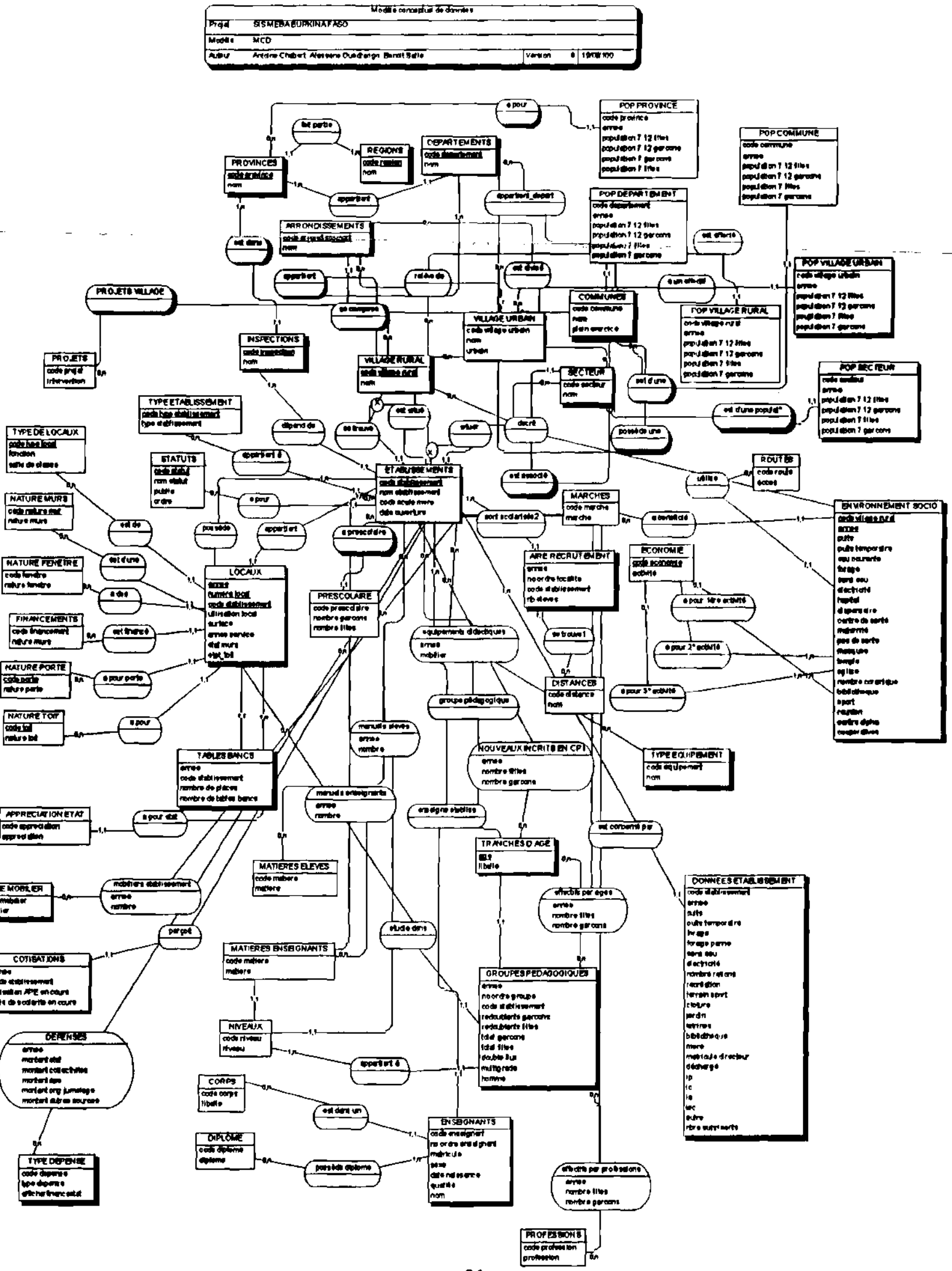
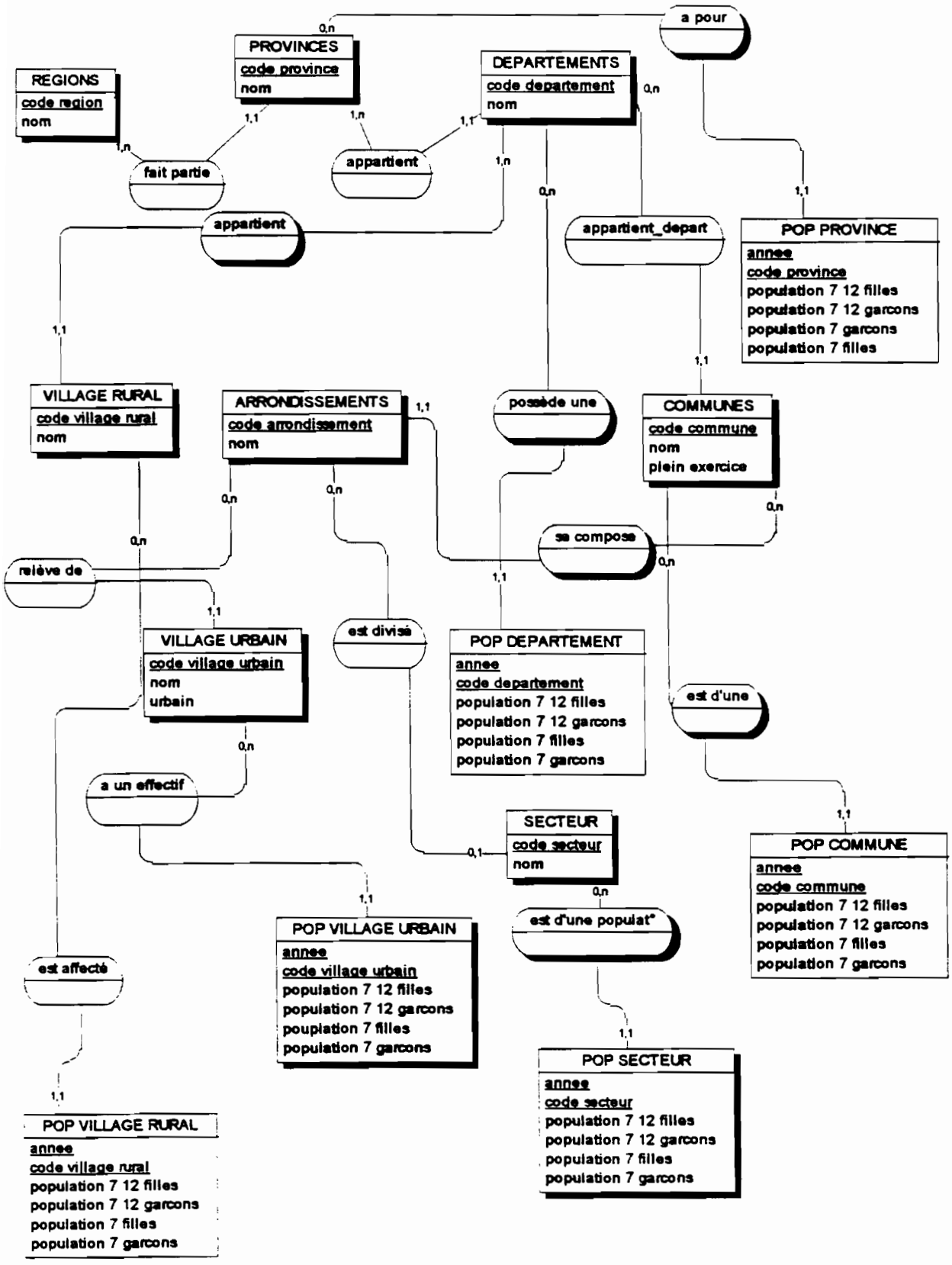


Figure n°49: Organigramme du PAEB

Annexe II : Le modèle conceptuel des données



Extrait des données sur la répartition administrative



Annexe III : Grammaire du langage XML en notation BNF

Document				
[1]	document	::=	prologue élément Divers*	

Ensemble de caractères					
[2]	Car	::=	#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]	/*	tout caractère Unicode, sauf les seizets d'indirection, FFFE et FFFF. */

Séparateurs				
[3]	S	::=	(#x20 | #x9 | #xD | #xA)+	

Noms et atomes nominaux			
[4]	CarNom	::=	Lettre | Chiffre | '.' | '-' | '_' | ':' | CarJonctif | ModificateurLettre
[5]	Nom	::=	(Lettre | '_' | ':') (CarNom)*
[6]	Noms	::=	Nom (S Nom)*
[7]	AtomeNml	::=	(CarNom)+
[8]	AtomesNmx	::=	AtomeNml (S AtomeNml)*

Littéraux			
[9]	ValeurEntité	::=	'"' ([^%&"] | AppelEP | Appel)* '"'
			|"'" ([^%&'] | AppelEP | Appel)* "'"
[10]	ValeurAtt	::=	'"' ([^<&"] | Appel)* '"'
			|"'" ([^<&'] | Appel)* "'"
[11]	LittéralSystème	::=	('"' [^"]* '"') |("'" [^']* "'")
[12]	IdPubLittéral	::=	'"' CarIdPub* '"' | "'" (CarIdPub - "'")* "'"
[13]	CarIdPub	::=	#x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$ %]

Données textuelles			
[14]	DonnéesTextuelles	::=	[^<&]* - ([^<&]* ']]>' [^<&]*)

Commentaires			
[15]	Commentaires	::=	'<!--' ((Car - '-') | ('-' (Car - '-')))* '-->'

Instructions de traitement			
[16]	IT	::=	'<?' CibleIT (S (Car* - (Car* '?>' Car*)))? '?>'
[17]	CibleIT	::=	Nom - (('X' | 'x') ('M' | 'm') ('L' | 'l'))

Sections CDATA				
[18]	SectionDT	::=	DébutDT DonnéesDT FinDT	
[CAPut!']	DébutDT	::=	<![CDATA['	
[20]	DonnéesDT	::=	(Car* - (Car* ']]>' Car*))	
[21]	FinDT	::=	']]>'	

Prologue					
[22]	prologue	::=	DéclXML? Divers* (déclTypeDoc Divers*)?		
[23]	DéclXML	::=	'<?xml' InfoVersion DéclCodage? DéclDocAuto? S? '?>'		
[24]	InfoVersion	::=	S 'version' Égal (' NumVersion ' | " NumVersion ")		
[25]	Égal	::=	S? '=' S?		
[26]	NumVersion	::=	([a-zA-Z0-9_.:] | '-')+		
[27]	Divers		Commentaires | IT | S		
[28]	déclTypeDoc	::=	'<!DOCTYPE' S Nom (S IdExterne)? S? ('[' (déclBalisage | AppelEP | S)* ']' S?)? '>'	[	CV : Type de l'élément racine]
[29]	déclBalisage	::=	déclÉlément | DéclListeAtt | DéclEntité | DéclNotation | IT | Commentaires	[	CV : Imbrication stricte des déclarations et des EP]
				[	CF : EP dans le sous-ensemble interne]

Sous-ensemble externe			
[30]	sousEnsembleExt	::=	DéclTexte? déclSousEnsembleExt
[31]	déclSousEnsembleExt	::=	(déclBalisage | sectConditionnelle | AppelEP | S)*

Déclaration de document autonome					
[32]	DéclDocAuto	::=	S 'standalone' Égal (("'" ('yes' | 'no') "'") | ('"' ('yes' | 'no') '"'))	[	CV : déclaration de document autonome]

Identification de Langue				
[33]	IdLang	::=	CodeLang ('-' SousCode)*	
[34]	CodeLang	::=	CodeISO639 | CodeIana | CodeUtil	
[35]	CodeISO639	::=	([a-z] | [A-Z]) ([a-z] | [A-Z])	
[36]	CodeIana	::=	('i' | 'I') '-' ([a-z] | [A-Z])+	
[37]	CodeUtil	::=	('x' | 'X') '-' ([a-z] | [A-Z])+	
[38]	SousCode	::=	([a-z] | [A-Z])+	

Élément					
[39]	élément	::=	BaliseÉlémVide		
			| BaliseO contenu BaliseF	[	CF : Correspondance de type d'élément]
				[	CV : Élément valide]

Balise ouvrante					
[40]	BaliseO	::=	'<' Nom (S Attribut)* S? '>'	[	CF : Spécif. unique de l'attribut]
[41]	Attribut	::=	Nom Égal ValeurAtt	[	CV : Type valeur de l'attribut]
				[	CF : Pas d'appel d'entite externe]
				[	CF : Pas de < dans les valeurs d'attribut]

Balise fermante				
[42]	BaliseF	::=	'</' Nom S? '>'	

Contenu des éléments					
[43]	contenu	::=	(élément | DonnéesTextuelles | Appel | SectionDT | IT | Commentaire)*		

Balises pour éléments vides					
[44]	BaliseÉlemVide	::=	'<' Nom (S Attribut)* S? '/>'	[	CF : Spécif. unique de l'attribut]

Déclaration de type d'élément					
[45]	déclÉlément	::=	'<!ELEMENT' S Nom S specContenu S? '>'	[	CV : Déclaration de type d'élément unique]
[46]	specContenu	::=	'EMPTY' | 'ANY' | Mixte | sousÉléments		

Modèle de contenu élémentaire					
[47]	sousÉléments	::=	(choix | séq) ('?' | '*' | '+')?		
[48]	pc	::=	(Nom | choix | séq) ('?' | '*' | '+')?		
[49]	choix	::=	'(' S? pc (S? '|' S? pc)* S? ')'	[	CV : Imbrication stricte des parenthèses dans EP]
[50]	séq	::=	'(' S? pc (S? ',' S? pc)* S? ')'	[	CV : Imbrication stricte des parenthèses dans EP]

Déclaration de contenu mixte					
[51]	Mixte	::=	'(' S? '#PCDATA' (S? '|' S? Nom)* S? ')*'		
			| '(' S? '#PCDATA' S? ')'	[	CV : Imbrication stricte des parenthèses dans EP]
				[	CV : Type unique]

Déclaration de liste d'attributs				
[52]	DéclListeAtt	::=	'<!ATTLIST' S Nom DéfAtt* S? '>'	
[53]	DéfAtt	::=	S Nom S TypeAtt S DéclDéfaut	

Types d'attribut					
[54]	TypeAtt	::=	TypeChaîne | TypeAtomique | TypeEnuméré		
[55]	TypeChaîne	::=	'CDATA'		
[56]	TypeAtomique	::=	'ID'	[	CV : ID]
				[	CV : Un seul ID par type d'élément]
				[	CV : Attribut ID par Défaut]
			| 'IDREF'	[	CV : IDREF]
			| 'IDREFS'	[	CV : IDREF]
			| 'ENTITY'	[	CV : Nom d'entité]
			| 'ENTITIES'	[	CV : Nom d'entité]
			| 'NMTOKEN'	[	CV : Atome nominal]
			| 'NMTOKENS'	[	CV : Atome nominal]

Types d'attributs énumérés					
[57]	TypeÉnuméré	::=	TypeNotation | Énumération		
[58]	TypeNotation	::=	'NOTATION' S '(' S? Nom (S? '|' S? Nom)* S? ')'	[	CV : Attributs de notation]
[59]	Énumération	::=	'(' S? AtomeNml (S? '|' S? AtomeNml)* S? ')'	[	CV : Énumération]

Valeurs par défaut des attributs					
[60]	DéclDéfaut	::=	'#REQUIRED' | '#IMPLIED'		
			| (('#FIXED' S)? ValeurAtt)	[	CV : Attribut obligatoire]
				[	CV : Valeur par défaut de l'attribut légale]
				[	CF : Pas de < dans valeurs d'attribut]
				[	CV : Valeur par défaut de l'attribut fixe]

Section conditionnelle					
[61]	SectConditionnelle	::=	sectInclude | sectIgnore		
[62]	sectInclude	::=	'<![' S? 'INCLUDE' S? '[' déclSousEnsembleExt ']]>'		
[63]	sectIgnore	::=	'<![' S? 'IGNORE' S? '[' contenuSectIgnore* ']]>'		
[64]	contenuSectIgnore	::=	Ignore ('<![' contenuSectIgnore ']]>' Ignore)*		
[65]	Ignore	::=	Car* - (Car* ('<![' | ']]>') Car*)		

Appel de caractère					
[66]	AppelCar	::=	'&#' [0-9]+ ';'		
			| '&#x' [0-9a-fA-F]+ ';'	[	CF : Caractère légal]

Appel d'entité					
[67]	Appel	::=	AppelEntité | AppelCar		
[68]	AppelEntité	::=	'&' Nom ';'	[	CF : Entité déclarée]
				[	CV : Entité déclarée]
				[	CF : Entité analysable]
				[	CF : Pas de récursion]
[69]	AppelEP	::=	'%' Nom ';'	[	CV : Entité déclarée]
				[	CF : Pas de récursion]
				[	CF : Dans la DTD]

Déclaration d'entité			
[70]	DéclEntité	::=	DéclEG | DéclEP
[71]	DéclEG	::=	'<!ENTITY' S Nom S DéfEntité S? '>'
[72]	DéclEP	::=	'<!ENTITY' S '%' S Nom S DéfEP S? '>'
[73]	DéfEntité	::=	ValeurEntité | (IdExterne DeclNdata?)
[74]	DéfEP	::=	ValeurEntité | IdExterne

Déclaration d'entité externe					
[75]	IdExterne	::=	'SYSTEM' S LittéralSystème		
			| 'PUBLIC' S IdPubLittéral S LittéralSystème		
[76]	DéclNdata	::=	S 'NDATA' S Nom	[	CV : Notation déclaree]

Déclaration de texte			
[77]	DéclTexte	::=	'<?xml' InfoVersion? DéclCodage S? '?>'

Entité analysable externe bien formée					
[78]	entAnalExt	::=	DéclTexte? contenu		
[79]	entParExt	::=	DéclTexte? déclSousEnsembleExt		

Déclaration de codage					
[80]	DéclCodage	::=	S 'encoding' Égal ('"' NomCodage '"' | "'" NomCodage "'")		
[81]	NomCodage	::=	[A-Za-z] ([A-Za-z0-9._] | '-')*	/*	Nom de codage ne contenant que des caractères latins */

Déclaration de notation			
[82]	DéclNotation	::=	'<!NOTATION' S Nom S (IdExterne | IdPublic) S? '>'
[83]	IdPublic	::=	'PUBLIC' S IdPubLittéral

Les caractères			
[84]	Lettre	::=	CarBase | Idéogramme
			[#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E] | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-

			#x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E | [#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]
[86]	Idéogramme	::=	[#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]
[87]	CarJonctif	::=	[#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED] | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43] | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48] | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A] | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A
[88]	Chiffre	::=	[#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9] | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F] | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF] | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F] | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]
[89]	Modificateur Lettre	::=	#x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46 | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]