Session conception : gestion des programmes dans l'atelier logiciel Alis

Jean Dalle Rive

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

le diplôme d'ingénieur

en

Informatique

par

Jean DALLE RIVE

Session Conception : GESTION DES PROGRAMMES DANS L'ATELIER LOGICIEL ALIS

SOUTENU LE :

JURY

Président : Professeur RANCHIN Y.

Membres : Professeur BOLLIET L.

M. CHUPIN J.C.

M. DOUSSY J.

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

le diplôme d'ingénieur

en

Informatique

par

Jean DALLE RIVE

Session Conception : GESTION DES PROGRAMMES DANS L'ATELIER LOGICIEL ALIS

SOUTENU LE :

JURY

Président : Professeur RANCHIN Y.

Membres : Professeur BOLLIET L.

M. CHUPIN J.C.

M. DOUSSY J.

Le travail présenté dans ce mémoire a été réalisé à BULL-SEMS dans l'équipe "Techniques Langages et Production de Programmes", dans le cadre du projet ALIS : Atelier Logiciel Intégré Sems.

Je tiens à remercier :

Monsieur Ranchin du Conservatoire National des Arts et Métiers qui m'a fait l'honneur de présider le jury.

Monsieur le professeur Bolliet qui m'a fait l'honneur d'accepter mon dossier de thèse, pour l'aide qu'il m'a apportée.

Monsieur J. Doussy responsable de l'équipe "Technique Langages et Production de Programmes" qui m'a acceuilli dans l'équipe Alis et a dirigé mes travaux.

Monsieur J. Estublier du laboratoire Imag de Grenoble pour la critique constructive qu'il a faite de ce mémoire.

Les personnes qui se sont occupées de la frappe et du tirage de ce mémoire.

Je tiens à remercier aussi mes camarades de l'équipe Alis qui ont participé à la réalisation pratique de cette étude.

- TABLE DES MATIERES -

1 INTRODUCTION 1.1

2 ATELIER LOGICIEL : POURQUOI FAIRE ? 2.1

2.1 AMELIORER LA PRODUCTIVITE DES EQUIPES 2.1

2.2 AUTOMATISER TOUT CE QUE EST REPETITIF 2.2

2.3 ASSURER LA QUALITE ET LA CONFORMITE DES LOGICIELS 2.2

2.4 PRODUIRE DANS DES CONDITIONS DE COUTS ET DE DELAIS CONNUS A L'AVANCE . . . 2.2

2.5 PRENDRE EN COMPTE L'EVOLUTION DES LOGICIELS 2.3

2.6 DIMINUER L'INDIVIDUALISATION DES TACHES 2.3

2.7 LES ATELIERS LOGICIELS 2.3

3 ATELIER LOGICIEL ALIS 3.1

3.1 LE CYCLE DE VIE DES PRODUITS SEMS 3.1

3.1.1 La PRE-ETUDE 3.1

3.1.2 La phase réalisation 3.2

3.1.3 La phase de suivi-maintenance 3.2

3.2 ARCHITECTURE ALIS-PRINCIPE GENERAL 3.2

4 LA MODULARITE DANS ALIS 4.1

4.1 LES AVANTAGES DE LA MODULARITE 4.1

4.2 LA MODULARITE DANS ALIS 4.2

4.3 LES "INCONVENIENTS" DE LA MODULARITE 4.5

5 ARCHITECTURE DU BIBLIOTHECAIRE D'ALIS 5.1

5.1 LES NIVEAUX 5.1

5.2 LES INFORMATIONS ASSOCIEES A CHAQUE NIVEAU 5.4

5.2.1 Au niveau projet 5.4

5.2.2 Au niveau produit 5.5

5.2.3 Au niveau application 5.5

5.3 GESTION DES VERSIONS DE DEVELOPPEMENT 5.7

6 CONCEPTION-SPECIFICATION 6.1

6.1 LES FONCTIONS DE L'EDITEUR DE DECOUPE 6.1

6.1.1 Découpe au niveau projet 6.1

6.1.1.1 Création de projet 6.1

6.1.1.2 Initialisation découpe du projet 6.2

6.1.1.3 Modification de la découpe du projet 6.2

6.1.1.4 Documentation découpe projet 6.2

6.1.2 Découpe au niveau produit 6.3
6.1.2.1 Initialisation de la découpe d'un produit 6.3
6.1.2.2 Modification de la découpe d'un produit 6.3
6.1.2.3 Documentation découpe produit 6.3
6.1.3 Découpe au niveau application : 6.4
6.1.4 Initialisation de la découpe 6.4
6.1.4.1 Modification de la découpe 6.5
6.1.4.2 Documentation découpe application 6.6
6.1.5 Les dialogues de la session conception 6.7
6.2 OBJETS LIEN ET LISTE DE COMPOSITION 6.10
6.2.1 Objet liste de composition 6.10
6.2.1.1 Niveau projet 6.10
6.2.1.2 Niveau produit 6.12
6.2.1.3 Niveau Application 6.14
6.2.2 Objet liens 6.16
6.2.2.4 Structure interne de l'objet lien 6.18
7 UTILISATION DES INFORMATIONS GEREES PAR LA SESSION CONCEPTION 7.1
7.1 LA SESSION PRODUCTION DE PROGRAMMES 7.1
7.1.1 Présentation générale 7.1
7.1.1.1 Compilateur et générateur-optimiseur 7.1
7.1.1.2 Editeur de liens et générateur d'images mémoires 7.4
7.1.2 Création du contexte de compilation d'un module 7.6
7.1.3 "Re"-compilations automatiques et re-édition des liens 7.6
7.1.4 Génération des commandes à l'éditeur des liens 7.7
7.1.5 Complexiмètre 7.7
7.1.6 Documentation 7.7
7.2 EXTENSIONS 7.8
7.2.1 Session mise au point - debugger symbolique 7.8
7.2.1.1 Présentation générale du debugger symbolique 7.8
7.2.2 Transport sur SM90 . . . . . 7.9
8 CONCLUSION 8.1
9 ANNEXES 9.1
9.1 ANNEXE A : COUT DU LOGICIEL 9.1
9.2 ANNEXE B - STRUCTURE D'UNE BIBLIOTHEQUE 9.6
9.3 ANNEXE C : INTERFACE OBJETS LISTE DE COMPOSITION ET OBJETS LIEN 9.9
9.3.1 Interfaces objets liste de composition 9.9

9.3.2 Interfaces objets lien 9.10
9.4 ANNEXE D : PRESENTATION DE QUELQUES ATELIERS 9.10
9.5 EM2 : UN ENVIRONNEMENT POUR MODULA2 . . . . . .9.10
9.6 LA BASE DE PROGRAMME ADELE 9.13
10 BIBLIOGRAPHIE 10.1

1 INTRODUCTION

Ce travail a été réalisé dans le cadre du projet Alis,Atelier Logiciel Intégré Sems. L'objectif de ce projet est de fournir aux équipes de développement de Sems un atelier pour la production de gros logiciels. Les travaux rapportés dans ce mémoire concernent :

- dans le premier chapitre, les objectifs généraux d'un atelier logiciel et les différentes approches possibles.

- dans le deuxième chapitre, la description de l'atelier Alis dans ses grands principes.

- dans le troisième chapitre, les concepts de modularité dans Alis.

- dans le quatrième chapitre, le bibliothécaire.

- dans le cinquième chapitre, la partie spécification/conception dans Alis, ainsi que la gestion des liens entre les entités manipulées.

- enfin le dernier chapitre présente les extensions prévues.

2 ATELIER LOGICIEL : POURQUOI FAIRE ?

La conception, la réalisation et la mise en oeuvre de systèmes informatiques de plus en plus complexes (par leur taille, les techniques utilisées, etc.) conduisent la profession informatique à transformer le travail artisanal en mise en place rapide du génie locigiel automatisé. La production "industrielle" des logiciels est aujourd'hui devenue une nécessité. C'est devant cette nécessité qu'est apparue l'idée d'atelier logiciel.

Le génie logiciel et les ateliers qui en sont les supports privilégiés de mise en oeuvre doivent répondre aux défis des années à venir, à savoir :

- améliorer la productivité des équipes de production.

- automatiser tout ce qui est répétitif, afin de permettre aux équipes de se concentrer sur les parties spécifiques.

- assurer la qualité et la conformité des logiciels.

- produire dans des conditions de coûts et de délais connus à l'avance.

- prendre en compte l'évolution des logiciels.

- diminuer l'individualisation des tâches.

2.1 AMELIORER LA PRODUCTIVITE DES EQUIPES

Les causes de la non productivité des équipes sont :

- Le manque de rigueur dans la définition et la conception des logiciels à fabriquer, d'où des itérations nombreuses et coûteuses pour les adaptations, les corrections d'erreurs.

- Le faible réemploi de composants logiciels déjà disponibles (non portabilité).

- Un environnement de travail trop artisanal,voire individualiste (méthode, outils, etc.).

- Une automatisation trop faible de processus pourtant généraux et répétitifs.

2.2 AUTOMATISER TOUT CE QUE EST REPETITIF

L'automatisation doit concerner tout ce qui est commun à plusieurs projets de logiciel; à savoir par exemple :

- La documentation qui doit être la même dans tous les projets.

- L'ordonnancement des tâches, les suivis de planning qui peuvent être pris en charge par des outils automatiques.

- La gestion des bibliothèques, des versions de composants qui sont difficiles à gérer sans l'assistance automatique d'outils .

2.3 ASSURER LA QUALITE ET LA CONFORMITE DES LOGICIELS

Les traditionnelles recettes ne couvraient qu'un aspect de la qualité : la fiabilité (le fameux "ça marche dans tous les cas"). Il faut en fait se préoccuper de la qualité d'un logiciel tout au long de son cycle de vie. On parle d'assurance qualité lorsqu'il s'agit de préconiser telle ou telle méthode d'analyse de programmation ou de test, et de contrôle qualité pour vérifier que le produit satisfait bien aux normes de qualité retenues.

Les critères de qualité retenus sont les suivants :

- qualité de réalisation (fiabilité et opérabilité).

- qualité de maintenance (fiabilité de modification, dépannage et recherche de pannes).

- qualité de portabilité (encapsulation des choix de réalisation, découpe en objets).

La qualité d'un logiciel doit pouvoir être mesurée à tous les niveaux : aussi bien au niveau d'un simple module (complexité d'un algorithme) qu'au niveau architecture globale d'un logiciel. Des outils spécifiques (du type analyseur de complexité) permettent de mesurer cette qualité.

Enfin, comme dans tous les domaines techniques, la qualité et la conformité ne vont pas sans une documentation qui corresponde aux différents états du logiciel et à ses évolutions passées ou à venir.

2.4 PRODUIRE DANS DES CONDITIONS DE COUTS ET DE DELAIS CONNUS A L'AVANCE

Pour garantir les coûts et maîtriser les délais il faut disposer de méthodes et de modèles d'évaluation qui prennent en compte la nature, la complexité des services à rendre, mais aussi qui intègrent les expériences passées. C'est pourquoi il faut disposer à tous les niveaux de réalisations d'outils de planification et de suivis (cf. Annexe A evolution des coûts du logiciel).

2.5 PRENDRE EN COMPTE L'EVOLUTION DES LOGICIELS

Un logiciel suit un cycle de vie et à ce titre est amené à connaitre des évolutions. A cet effet les ateliers logiciels doivent fournir des outils qui permettent de prévoir et de constater les conséquences des évolutions envisagées.

2.6 DIMINUER L'INDIVIDUALISATION DES TACHES

Il arrive souvent qu'une tâche soit confiée à un seul individu qui doit la mener du début à la fin. L'absence provisoire ou définitive d'un tel individu a souvent des conséquences désagréables (formation d'une nouvelle personne sur le produit, ce qui peut prendre beaucoup de temps). Il faut donc banaliser le couple logiciel à produire/individu producteur. Pour cela l'atelier doit être capable de fournir des informations précises sur l'état d'un logiciel : par exemple l'architecture globale du logiciel, l'état d'avancement de ce logiciel.

2.7 LES ATELIERS LOGICIELS

Les méthodes traditionnelles de fabrication de logiciel se réduisent à un système d'exploitation offrant des outils classiques (compilateurs, éditeur, etc.). Chaque outil utilise un formalisme propre tant du point de vue interface usager, que du point de vue structure de données manipulées. De plus chaque outil ignore les autres outils et l'environnement de l'objet manipulé. Citons par exemple UNIX qui offre un ensemble d'outils regroupés dans l'environnement PWB [PWB 77,FEL 79].

Par opposition un atelier logiciel, dans l'état actuel des besoins [STO 80], comporte un ensemble d'outils, dédiés à chaque phase du cycle de vie du logiciel, intégrés et construits autour d'une "base de données" logicielle qui constitue le noyau de l'atelier et le moyen de communication entre les outils[KRA 81,FOI 81,CAD 81].

Pour satisfaire les besoins cités précédemment, certains ateliers [CAD 81,FOI 81] Utilisent des bases de donneés existantes. D'autres [SAL 80] Préfèrent développer des bases de données spécialisées dans le génie logiciel. Dans les deux cas le système de gestion de la base de données logicielle doit connaître le modèle et la méthode de développement du logiciel :

> L'atelier doit connaitre les différentes entités à gérer , leurs propriétés et les relations qui les relient. Une base de données spécialisée , où le modèle des logiciels supportés est figé, est plus performante en étant plus adaptée à cette structure, alors qu'une base de données générale permet des modèles divers , au détriment de la performance. Dans tous les ateliers , qu'ils utilisent ou non un SGBD la base de données logicielle est donc primordiale.

Plusieurs autres tendances se font remarquer :

- certains ateliers ont été développés pour une méthode de travail particulière [SNO 81,WIL 81].

- d'autres sont dédiés à un langage [EM2 83,SCH 82]. Le développement des ateliers logiciels est souvent allé de pair avec le développement de langage exprimant la decomposition modulaire d'un logiciel en entités ayant des relations entre

elles : notion de module comportant une partie définition (ou spécification) et une partie réalisation, expression de la visibilité entre ces entités par des directives Import et Export par exemple.

L'atelier logiciel Alis :

- offre un ensemble d'outils intégrés autour d'une base de données logicielle spécialisée.

- est dédié à un langage modulaire.

Le chapitre suivant présente les principes et l'organisation générale d'Alis.

3 ATELIER LOGICIEL ALIS

ALIS (Atelier Logiciel Intégré Sems) est la réponse de Sems au développement de ses propres systèmes. Le contexte de cet atelier est donc le suivant : constructeur de mini-ordinateurs ayant à écrire et industrialiser de gros logiciels (système d'exploitation par exemple). L 'atelier logiciel Alis est bâti autour du cycle de vie des produits et intervient dans chacune des étapes de ce cycle.

3.1 LE CYCLE DE VIE DES PRODUITS SEMS

Ce cycle est tout à fait classique. On peut le caractériser par la longue durée de vie des logiciels et par les évolutions qui sont réalisées en permanence pour améliorer leur qualité et les adapter aux besoins du marché [DEL 81] .

Le cycle de vie comprend trois grandes phases :

- la pré-étude
- la réalisation
- le suivi.

3.1.1 La PRE-ETUDE

Elle débute par une étude marketting qui se concrétise par une "Spécification de besoins" qui est un document "produit".

A partir de ce document la direction technique élabore un document appelé "Spécification d'Objectif" pour un produit qui répond aux besoins exprimés. Un certain nombre de revues permettent de valider la spécification d'objectif. C'est un document technique.

Dans ce document figurent :

- les fonctionnalités du produit
- l'environnement système et matériel
- l'architecture du produit (technique et commerciale)
- la documentation à réaliser
- les étapes de réalisation
- les performances attendues

3.1.2 La phase réalisation

Cette phase se passe au sein de la direction technique. Elle se décompose en :

- étude ou conception
- programmation et tests
- qualification
- intégration.

Cette phase est complètement prise en charge par une équipe qui fournit à la fin le produit commandé. Le document essentiel issu de la phase conception est appelé "Spécification de Définition", il valide la conception. Ce document comprend :

- La définition complète et précise des interfaces externes et internes du produit.
- Les solutions techniques.
- La définition complète de l'environnement logiciel et matériel.

3.1.3 La phase de suivi-maintenance

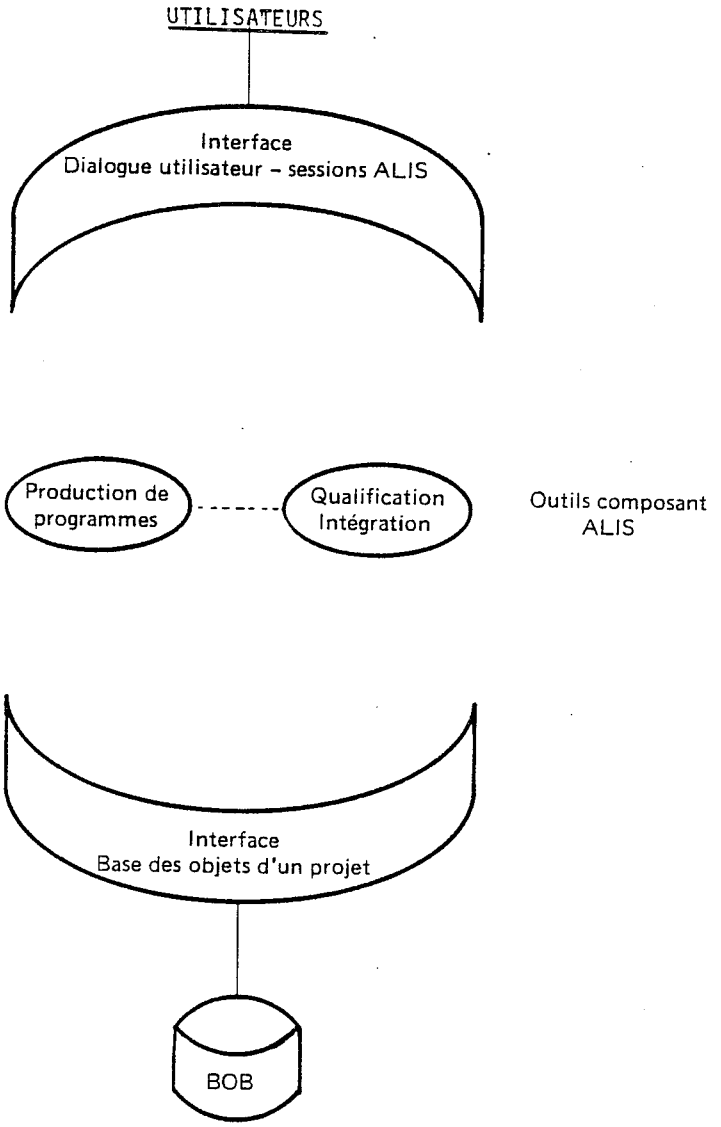
Cette phase est consacrée :

- à la livraison des produits.
- à la maintenance des produits (essentiellement correction des anomalies).
- aux évolutions des produits mineures ou importantes.

Au cours de cette phase il est souvent indispensable de sortir de nouvelles versions d'un produit; à chaque fois, on passera par une phase qualification-intégration.

3.2 ARCHITECTURE ALIS-PRINCIPE GENERAL

Alis est avant tout une structure d'accueil pour tous les outils qui s'intègrent dans l'atelier [DOU 82] .



Organisation de l'atelier ALIS :

- Dialogue avec les utilisateurs, notion de sessions
- Outils regroupés en sessions spécialisées
- Base des objets : stockage, accès, liens entre objets.

Organisation de l'atelier :

Dialogue avec les utilisateurs :

Le dialogue doit être agréable. Alis devant être intégré dans un poste de travail personnalisé, le dialogue doit tenir compte des aspects :

- Terminal intelligent
- Ecran graphique.

Les sessions spécialisées :

Etant donné le cycle de vie d'un produit, l'atelier logiciel doit servir de guide à l'utilisateur dans sa réalisation.

Un produit par exemple est découpé en différentes applications dont la réalisation est confiée à plusieurs personnes ou équipes. Les opérations à faire sur le produit doivent respecter le cycle de vie et l'atelier doit garantir leur bonne succession.

Une session est donc dédiée à une phase du développement d'un produit :

- elle fournit une interface "commande" adaptée aux traitements permis dans cette phase à l'aide d'un dialogue agréable et pédagogique.
- Elle contrôle les accès aux différents objets gérés par l'atelier en fonction de la découpe du produit.
- Elle assure la cohérence et la correction des opérations par des messages prévenant des modifications qui ont eu lieu, des recommandations ou des obligations.

On trouve cinq sessions principales dans Alis :

- Conception/spécification
- réalisation et tests
- qualification et intégration
- communication
- maintenance et suivi.

Dans chacune de ces sessions figurent des outils spécialisés, par exemple un outil contrôle qualité dans la session qualification-maintenance, tous les outils de production de programmes dans la session réalisation.

Le bibliothécaire :

IL gère l'ensemble des objets introduits et créés dans l'atelier à savoir par exemple : les modules dans leurs différentes formes, source, table des symboles, binaires..., ou encore les liens de visibilité et de dépendance entre les applications/modules d'un produit. C'est donc un magasinier.

D'autre part il règle l'accès à ces objets en respectant l'architecture d'un système en accord avec le type de session en cours. (Dans la session réalisation/test, qui fait appel aux outils de production de programmes, on peut accéder aux différentes formes que peut prendre un module (forme

source, forme Li, forme BT...)].

Enfin le bibliothécaire gère l'ensemble des liens qui représentent l'architecture du projet :

- liens entre modules (cf. Modularité)

- liens entre applications (cf. Modularité)

Cette gestion de liens permet d'avoir des outils de suivi du développement de produit, de documentation (ordre de compilation, recompilations automatiques...).

Par ailleurs le bibliothécaire gère plusieurs versions des objets gardés en bibliothèque. Cette gestion de version permet de conserver plusieurs fois le même objet. Elle est détaillée au paragraphe Architecture Alis.

4 LA MODULARITE DANS ALIS

Alis offre, à travers le langage PASCAL/S, des mécanismes permettant le découpage modulaire des logiciels.

4.1 LES AVANTAGES DE LA MODULARITE

Une grande partie des problèmes liés au développement des logiciels provient de la difficulté que l'on a à maîtriser la complexité des objets construits. Ainsi depuis plusieurs années, l'accent est mis sur le découpage d'un logiciel en unités logiques de taille plus petite et sur la définition des liens logiques entre ces différentes unités. D'où l'apparition des termes de programmation modulaire, module[LOY 81,CHV 82]. Les avantages d'une telle méthode sont les suivants :

- Le découpage d'un logiciel en unités plus petites reliées entre elles par des mécanismes d'interface est essentiellement un découpage logique, indépendant des choix de réalisation des diverses fonctions. Cela permet de mieux comprendre la structure d'un logiciel et de mieux cerner la fonctionnalité de chaque unité.

- La réalisation de chaque unité peut être faite indépendamment des réalisations des autres unités, chaque réalisation devant satisfaire les fonctionnalités de l'unité qu'elle réalise.

- Les liens entre les unités étant définis, en cas d'erreur dans une unité, il est aisé d'en mesurer les conséquences sur les autres unités. On peut ainsi, remplacer une réalisation d'une unité par une autre, qui satisfait les mêmes fonctionnalités, corrigeant l'erreur ou plus performante. De même on peut envisager d'avoir pour une unité une réalisation standard et une réalisation "maintenance" permettant la recherche d'erreurs.

- Le découpage en unités logiques doit permettre la réutilisation du logiciel, chaque unité étant caractérisée par ses spécifications et non par sa réalisation.

Il faut donc disposer de langages de programmation permettant d'exprimer cette modularité. Ainsi tout un ensemble de langages est apparu permettant d'écrire une application non pas en un seul programme mais en plusieurs unités souvent appelées modules. Citons les plus connus :

- Modula2 (cf. Annexe E)
- Mesa
- Legos [LOY 81]
- Euclid
- Clu
- Ada etc.

4.2 LA MODULARITE DANS ALIS

La modularité dans ALIS est inspirée des concepts développés dans LEGOS ou MESA . Le langage est un sur-ensemble du Pascal et s'appelle PASCAL/S [COQ 80]. On distingue les entités suivantes :

Module : : C'est l'unité de programmation

Un module comporte deux parties :

- une partie définition : qui définit un ensemble d'objets (constantes, types, variables, procédures et fonctions du langage) qui sont visibles de l'extérieur du module.

- une partie réalisation : qui est un programme standard du langage contenant des objets privés non visibles à l'extérieur et qui réalisent la partie définition.

Application : les modules peuvent être regroupés pour former une application. Associé à une application on a le mécanisme d'interface . Une interface sur une application se décrit par un module Interface.

Un module interface est en fait un module définition de l'ensemble de modules compris dans l'application. Dans un module interface on trouve des constantes, des types, des procédures et des fonctions issus des modules définition de l'application. On peut construire plusieurs interfaces sur une application, ce qui permet de donner plusieurs visibilités d'une même application. Par ailleurs les types peuvent être partiellement ou totalement cachés, les variables peuvent être en lecture uniquement.

Les liens de visibilité entre les entités dans Alis s'expriment sous deux formes par des ordres Import et Interface :

IMPORT : La partie définition d'un module définit l'ensemble des objets susceptibles d'être vus par d'autres modules de la même application, c'est-à-dire l'ensemble des objets exportables. Un module qui veut "voir" des objets déclarés comme exportables par un autre module, doit faire référence à la partie définition de cet autre module. On dit qu'il importe cette partie définition . Un module peut vouloir importer un ou plusieurs modules définition pour deux raisons.

- utiliser des symboles (constantes ou types) définis dans ces modules. C'est en général le cas d'un module définition important un autre module.

- utiliser les objets définis par ces modules. C'est le cas d'un module réalisation qui importe un module définition pour accéder aux procédures, fonctions qui y sont définies.

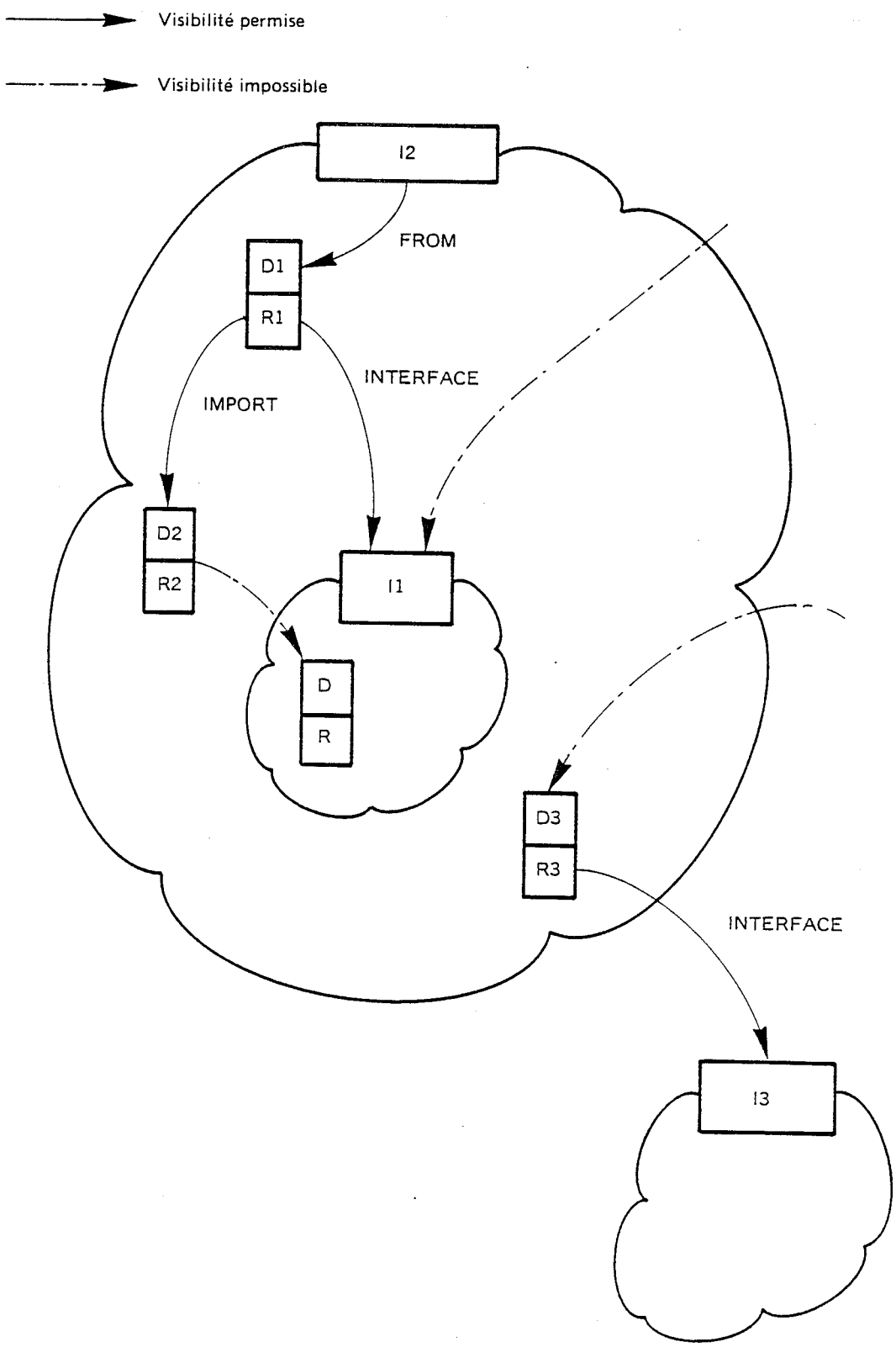
Interface : Le mécanisme est le même que pour Import. Une Interface définit cependant les objets qui sont visibles de l'extérieur d'une application par des modules d'autres applications. C'est donc le moyen de communication entre deux applications.

Les directives Import et Interface définissent donc une relation entre les modules. Cette relation n'est pas transitive. Ainsi si un module M1 importe un module Définition M2 qui lui même importe un module définition M3, alors M1 n'importe pas M3. Il en est de même pour la relation définie par Interface. (Si M1 veut voir les objets définis dans M3, il doit explicitement importer M3). Par ailleurs l'utilisation des directives

Import et Interface implique un ordre de compilation des modules : si un module M1 importe (ou interface) un module définition M2 (un module interface I1), le module définition M2 (interface I1) doit avoir été compilé auparavant.

Il faut noter que les cycles entre les modules définition ou les modules interface sont impossibles, puisqu'on ne pourrait alors effectuer de compilation. Dans la plus part des langages les liens de visibilité s'expriment directement dans le langage [LOY 81].Dans Alis les directives Import et Interface associées à chaque module ne figurent pas obligatoirement dans le langage. Elles sont décrites par l'utilisateur dans la session Conception : cette session permet d'exprimer la découpe du logiciel suivant les entités : Projet-Produit-Application-Module. Cette session est décrite dans le cinquième chapitre.

Le schéma suivant résume les notions de modularité.


Visibilité permise
Visibilité impossible
I2
FROM
D1
R1
INTERFACE
IMPORT
D2
R2
I1
D
R
D3
R3
INTERFACE
I3

4.3 LES "INCONVENIENTS" DE LA MODULARITE

Les inconvénients de la modularité proviennent de l'interdépendance des modules et applications entre eux. La modification de la décomposition modulaire d'un logiciel doit être possible à tout moment. Or ce type de modification peut transformer radicalement le graphe des dépendances et entraîner par exemple une cascade de recompilation. L'utilisateur doit donc être averti des conséquences de ces modifications (en obtenant la liste des modules à recompiler), et, le cas échéant être assisté dans ces opérations de modification. Ceci ne peut être fait que si la structure du logiciel en application/modules est connue du système de production . C'est pourquoi une session d'Alis, dite session conception, permet à l'utilisateur de décrire, modifier la découpe modulaire d'un logiciel. La session conception s'appuie sur un outil appelé éditeur de découpe. Cet outil travaille avec le bibliothécaire et permet de décrire la découpe du logiciel produit dans le cadre de l'atelier Alis.

Dans le chapitre suivant est precisée l'organisation générale du bibliothécaire ,ensuite la session conception et l'éditeur de découpe de cette session, enfin sont décrites les utilisations des informations gérées par cette session par les outils d'autres sessions.

5 ARCHITECTURE DU BIBLIOTHECAIRE D'ALIS

L'ensemble des objets manipulés par les outils de l'atelier ALIS,parfois de natures très différentes,sont gérés par un bibliothécaire. Pour ranger les gros objets (binaires,tables des symboles ...) le bibliothécaire utilise une technique classique qui s'apparente à des bibliothèques spécialisées [cf. Annexe B]. Pour la gestion des liens entre les entités d'Alis deux objets spécifiques ont été développés.

Le bibliothecaire est organisé suivant trois niveaux.

5.1 LES NIVEAUX

5.1.0.1 niveau projet

Ce niveau correspond à un ou plusieurs produits destinés à des clients. Alis est lui-même par exemple un projet : il fournit le produit atelier logiciel mais aussi des produits comme un compilateur FORTRAN 77 qui peut être utilisé hors du contexte atelier logiciel. Un projet correspond à un groupe de travail organisé autour d'un responsable de projet.

Projet = ensemble de produits

5.1.0.2 Niveau Produit

Chaque produit du projet regroupe des activités voisines.

Exemple : dans le projet Alis, il y a un produit compilateur, un produit éditeur de liens, un produit bibliothécaire, etc.

Un produit s'exprime en terme d'applications. La notion d'application a été définie au paragraphe "Modularité dans Alis".

Produit = ensemble d'applications

5.1.0.3 Niveau application

C'est donc un ensemble de modules. Une application réalise un ensemble de fonctionnalités exprimées pour une ou plusieurs interfaces. C'est à ce niveau qu'a lieu réellement la programmation.

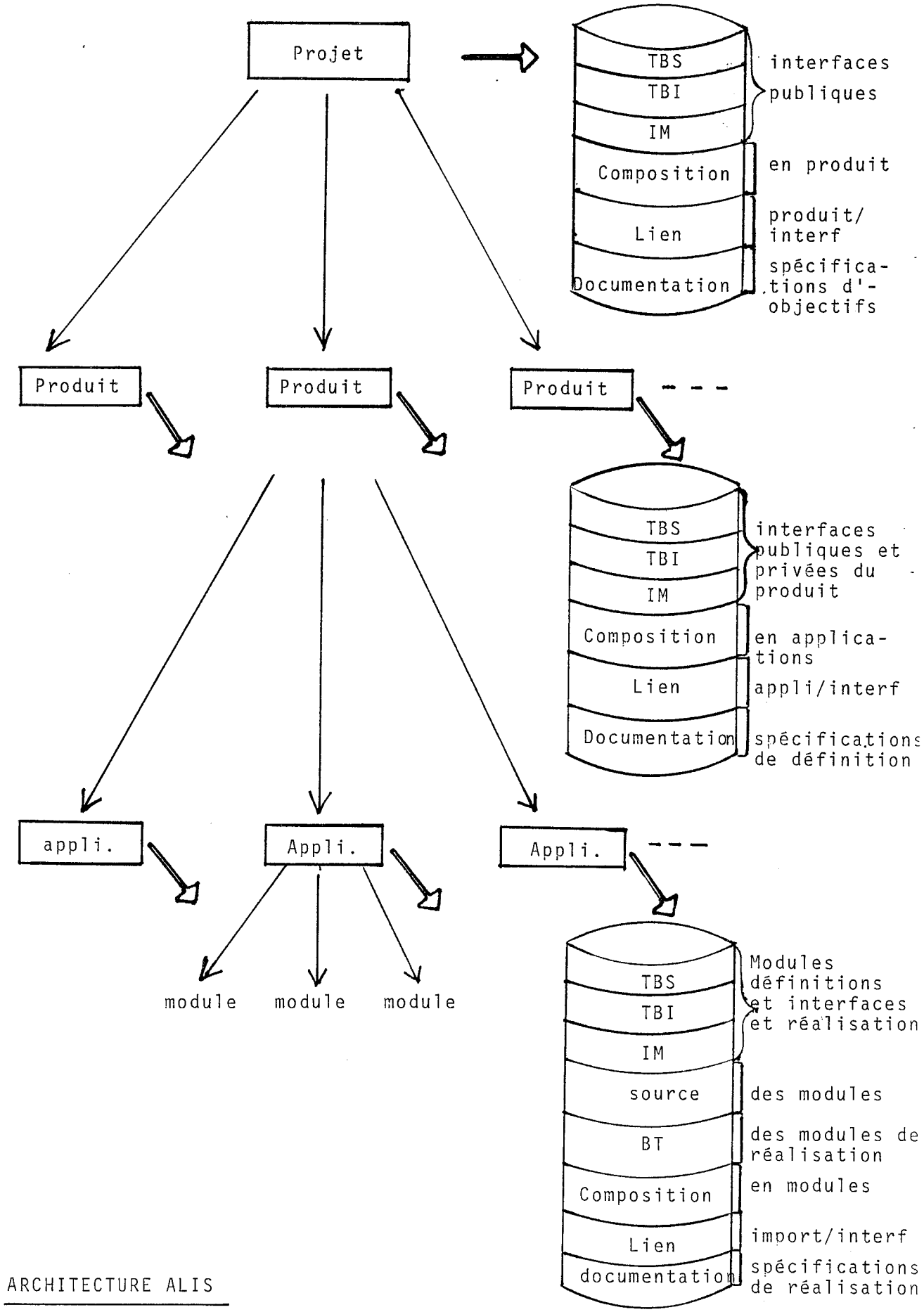
Application = ensemble de modules

Cette découpe en trois niveaux correspondant en fait à une démarche "TOP-Down" [AL 83] :

Le niveau projet exprime la découpe globale du projet. Ce niveau projet reflète le document spécification d'objectifs, document élaboré lors de la phase de pré-étude (cf. Paragraphe sur le cycle de vie des produits).

Ensuite chaque produit (il peut y en avoir un seul dans un projet) est découpé en applications, puis chaque application en modules. Les niveaux produit et application correspondent à la phase de réalisation.

L'architecture d'Alis peut donc se schématiser par un arbre, à chaque niveau étant attachées des informations de natures différentes.



ARCHITECTURE ALIS

5.2 LES INFORMATIONS ASSOCIEES A CHAQUE NIVEAU

5.2.1 Au niveau projet

On trouve à ce niveau :

- La liste de composition du projet : c'est-à-dire les produits développés dans le cadre du projet.

  Pour chaque produit figure :

  - son identification et des informations de gestion.

  - les interfaces (au sens Alis) éventuelles de ce produit. Ces interfaces sont dites publiques, car utilisables par tout produit du projet. Les noms de ces interfaces sont uniques pour le projet.

- La documentation qui comprend :

  - le document spécification d'objectif.

  - les documents décrivant les produits et les interfaces de ces produits.

- Un ensemble de bibliothèques, gérées par le bibliothécaire, qui contiennent :

  - les tables des symboles des interfaces publiques[cf. Chapitre production de programmes].

  - les tables d'interface des interfaces publiques[cf. Chapitre production de programmes].

  - les images mémoire des applications sur lesquelles sont construites les interfaces publiques[cf. Chapitre production de programmes].

- un objet lien : qui mémorise la relation Produits/Interfaces publiques.

Remarques :

Au niveau projet il faut distinguer deux classes d'interfaces :

- celles issues des produits développés dans le cadre du projet et dont le projet a la responsabilité.

- celles issues d'autres projets et qui sont donc simplement utilisées pour le développement du projet. Ces interfaces sont amenées lors de la phase d'initialisation du projet.

5.2.2 Au niveau produit

On trouve à ce niveau :

- la liste de composition du produit : c'est-à-dire les applications qui réalisent le produit. Pour chaque application figure :

  - son identification et des informations de gestion

  - sa ou ses interfaces associées.

On trouve donc obligatoirement les interfaces publiques déclarées au niveau projet. Les autres interfaces déclarées au niveau produit correspondent à des interfaces privées au produit. Ces interfaces privées ne sont donc visibles que par les applications du produit.

- La documentation qui comprend :

  - le document spécification de définition (cf. Paragraphe sur le cycle de vie du projet).

  - les documents décrivant les applications et leurs interfaces associées.

- un ensemble de bibliothèques, gérées par le bibliothécaire, qui contiennent :

  - Les tables des symboles des interfaces publiques et privées du produit.

  - Les tables d'interface des interfaces publiques et privées du produit.

  - Les images mémoires des applications sur lesquelles sont construites les interfaces publiques et privées du produit.

- un objet lien : qui mémorise la relation Applications / Interfaces publiques et privées.

5.2.3 Au niveau application

On trouve au niveau application :

- la liste de composition de l'application : c'est-à-dire les modules réalisant l'application.

  Pour chaque module figure :

  - son identification et des informations de gestion.

  - les liens avec les autres modules (Import) et les autres applications (Interface).

- La documentation qui comprend :

  - un document spécification de l'application qui en décrit les fonctionnalités.

  - un document décrivant chaque module de l'application.

- Un ensemble de bibliothèques, gérées par le bibliothécaire qui contiennent :

- les sources des modules.

- les tables des symboles des modules définition, réalisation et interface de l'application.

- les binaires [BT-Alis] Des modules réalisation.

- les images mémoire de l'application.

- le langage intermédiaire (LI) issu de la compilation des modules réalisations. Il est utilisé pour la mise au point avec le débugger symbolique [cf. Chapitre production de programmes].

- un objet lien qui mémorise les relations Import et Interface entre les modules.

Conclusion : L'architecture d'Alis reflète la méthode d'analyse descendante qui sera celle de l'utilisateur d'Alis. Le chapitre suivant décrit les outils de la session conception qui permettront à l'utilisateur de décrire son logiciel en termes de projet-produit-application-module et liens de visibilité entre ces différents composants.

5.3 GESTION DES VERSIONS DE DEVELOPPEMENT

Au cours du développement d'un logiciel on est amené à produire plusieurs versions des objets manipulés (source, table des symboles...). Alis offre une gestion des versions de développement.

5.3.0.1 Au niveau projet

Les objets communs à tout le projet figurent en deux versions :

- une version en cours : cette version est validée et utilisée par tous les réalisateurs
- une version "à valider" : contient les versions nouvelles des objets livrés du niveau produit, au niveau projet. C'est le chef de projet qui décide du passage de la version 'à valider' à celle en cours. A partir des informations de lien, figurant à ce niveau, il est capable d'avertir les responsables des produits concernés par cette décision.

5.3.0.2 Au niveau produit

Chaque produit peut figurer en plusieurs versions (V0 d'un compilateur, V1 de ce compilateur, etc...).

Créer une nouvelle version d'un produit signifie que les objets sont dupliqués : bibliothèques des tables des symboles, liste de composition, lien, images mémoire exécutables, documentation. Cette duplication se fait en prenant la version courante des objets de toutes les applications (source par exemple) et en les dupliquant dans la nouvelle version du produit (voir version niveau application).

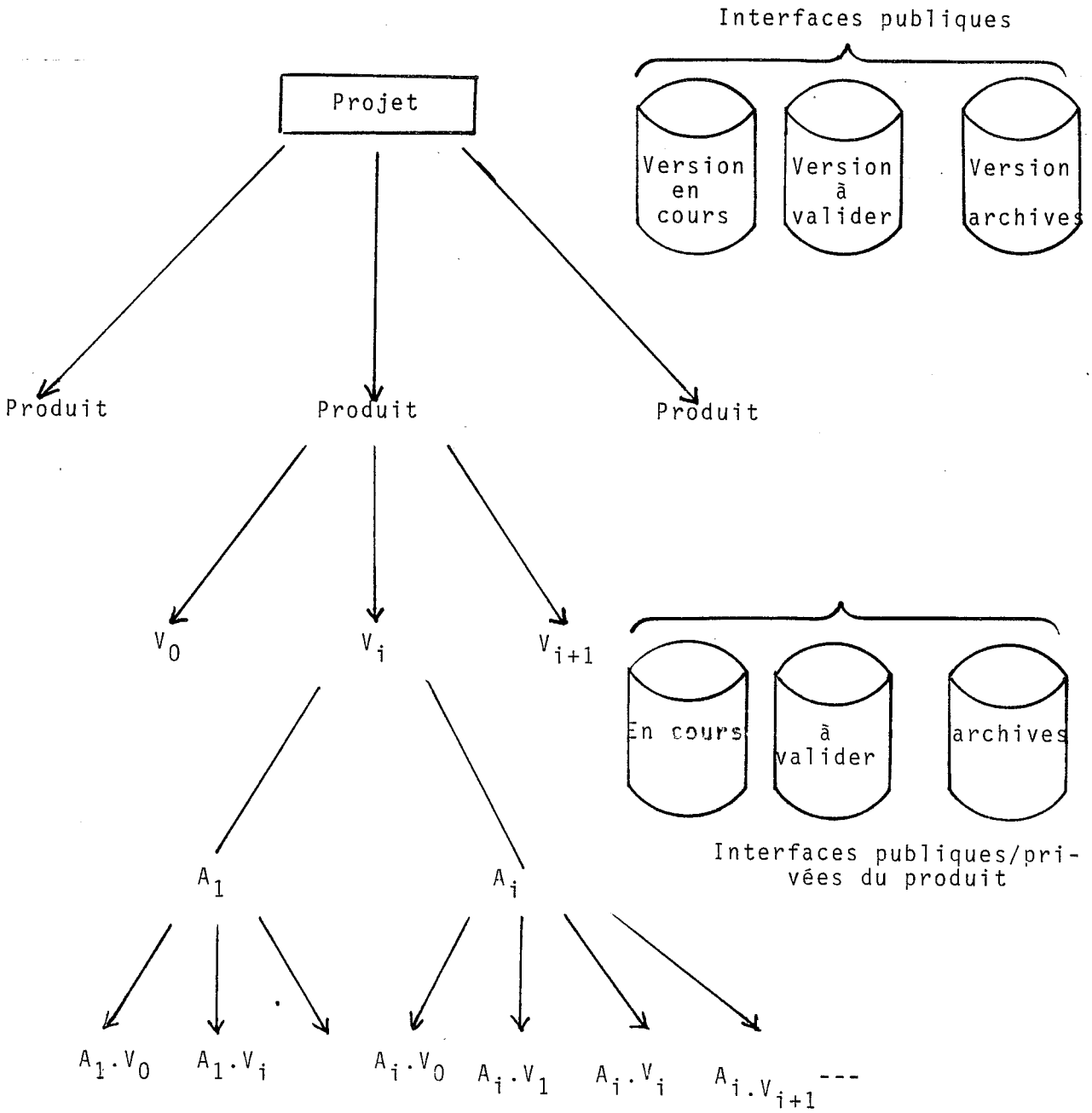
5.3.0.3 Au niveau application

Les informations gérées au niveau application sont toutes regroupées par nature (source, BT, ...) dans des bibliothèques dont l'accès est normalisé. Ces bibliothèques sont organisées sous la forme :

- des objets eux-mêmes.
- de répertoires : qui contiennent l'identification des objets.

Chaque répertoire correspond à une version de l'application. Créer une nouvelle version d'une application consiste à dupliquer le dernier répertoire de chacune des bibliothèques. Ceci permet de ne pas avoir duplication des objets eux-mêmes, puisque plusieurs éléments de répertoires distincts peuvent pointer le même objet. Par défaut le réalisateur de l'application travaille sur le répertoire de niveau le plus élevé. S'il désire travailler sur une autre version de l'application il doit préciser le numéro de cette version

Les schémas figurant en annexe B expliquent le fonctionnement des mécanismes répertoire/objet et de changement de version.



Gestion des versions- Principe

6 CONCEPTION-SPECIFICATION

Introduction

La conception-spécification se fait dans le cadre d'une session spécialisée dite session conception.

Cette session permet de décrire la découpe d'un projet suivant les niveaux : projet-produit-application-module. Cette decoupe est memorisée par l'outil éditeur de découpe dans deux objets gérés par le bibliothecaire . Dans plusieurs ateliers ou environnement de programmation [CAD 81,EM2 83,FOI 80] Les relations entre les entités manipulées sont gérées par une base de données[cf. Annexe D]. Dans Alis ces relations sont gérées par deux objets appelés liste de composition et objet lien. Ces objets sont décrits dans des interfaces [cf. Annexe C]. Ils sont crées et mis à jour par l'éditeur de découpe de la session conception. Ils sont consultés par les outils des autres sessions.

Les choix suivants ont été faits :

- L'outil de découpe est un éditeur de découpe . Il n'y a pas de langage, avec une syntaxe précise, de découpe. L'outil est interactif.

- La découpe d'un projet ne pouvant se faire en une seule fois de nombreux retours arrière sont nécessaires, à tous les niveaux.

- L'éditeur de découpe est adapté à chaque niveau d'Alis : les fonctions de cet éditeur sont différentes au niveau projet et application par exemple, ce qui permet d'avoir une découpe progressive dans le temps.

6.1 LES FONCTIONS DE L'EDITEUR DE DECOUPE

Ces fonctions sont spécifiques de chaque niveau de l'atelier.

6.1.1 Découpe au niveau projet

L'éditeur de découpe offre les fonctions :

- création de projet
- initialisation découpe du projet en termes de produit
- modification de la découpe du projet
- documentation.

6.1.1.1 Création de projet

C'est la première fonction à effectuer. L'utilisateur donne le nom du projet à initialiser. Les structures d'accueil des données, gérées au niveau projet, sont créées et initialisées (bibliotheques, liste de composition, objet lien...).

6.1.1.2 Initialisation découpe du projet

Cette opération, après celle de création, permet de donner une première découpe du projet en produits; elle consiste donc à :

- donner le nom des produits.

- éventuellement décrire les interfaces publiques construites sur chaque produit. Les noms des interfaces publiques sont uniques pour tout le projet . Un contrôle est effectué permettant d'assurer cette unicité.

- donner la liste des interfaces externes (provenant d'autres projets). A partir de ces informations on initialise les répertoires des bibliothèques situées au niveau projet (cf. Annexe B structure d'une bibliothèque Alis), ainsi que l'objet lien (cf. Gestion des liens).

6.1.1.3 Modification de la découpe du projet

Modification signifie :

- soit créer un nouveau produit et ses interfaces éventuelles

- soit supprimer un produit

- soit créer une nouvelle interface publique sur un produit

- soit supprimer une interface sur un produit.

La suppression d'un produit est une opération très rarement effectuée. Cependant cette opération est possible et a pour conséquence la suppression d'éventuelles interfaces publiques utilisées par d'autres produits du projet. Les liens de dépendance produit/interface publiques permettent de connaître les produits concernés par ces suppressions éventuelles.

L'opération de suppression d'une interface est similaire. A partir des informations de dépendance les produits concernés sont identifiés.

6.1.1.4 Documentation découpe projet

Cette fonction permet d'obtenir diverses informations sur le projet :

- Documentation globale : fournit à l'utilisateur la découpe globale du projet en produits, avec pour chaque produit les interfaces publiques associées,et les interfaces publiques qu'il utilise.

- Liste des produits : cette fonction fournit la liste des produits figurant dans le projet.

- Liste des interfaces publiques : cette fonction, ainsi que la précédente, permet à un utilisateur de connaître les noms des interfaces publiques (rappel : ces noms sont uniques pour le projet).

- Lien : l'utilisateur peut connaître à tout moment quel(s) produit(s) utilise(nt) l'interface publique de nom xxx. De même il peut connaître quelles sont les interfaces publiques utilisées pour le produit de nom xxx.

- Liste des interfaces figurant au niveau projet.

6.1.2 Découpe au niveau produit

Rappel : un produit s'exprime en termes d'applications Alis[cf. Modularité dans Alis].

Les fonctions de l'éditeur sont les suivantes :

- initialisation de la découpe d'un produit en termes d'applications
- modification de la découpe
- documentation

6.1.2.1 Initialisation de la découpe d'un produit

Cette opération permet de donner une première découpe du produit en applications; elle consiste donc à donner :

- le nom des applications
- pour chaque application les noms des interfaces privées ou publiques associées. Les noms des interfaces publiques doivent avoir été déclarés au niveau projet. Les noms des interfaces privées sont uniques pour un produit[mais dans deux produits différents il peut y avoir des interfaces privées de même nom].

A partir de ces informations on initialise les répertoires des bibliothèques situées au niveau produit (cf. Annexe B Structure d'une bibliothèque Alis), ainsi que l'objet lien.

6.1.2.2 Modification de la découpe d'un produit

Les opérations de modification possibles sont les suivantes :

- déclaration d'une nouvelle application et de ses interfaces publiques ou privées éventuelles ,en effectuant le contrôle d'unicité des noms d'interfaces publiques.
- suppression d'une application
- ajout d'une interface privée ou publique sur une application
- suppression d'une interface privée ou publique sur une application.

Ces opérations mettent à jour la liste de composition du produit ainsi que l'objet lien niveau produit.

La suppression d'une interface publique n'est possible que si celle-ci a été supprimée au niveau projet.

La suppression d'une application entraîne la suppression de toutes les interfaces associées.

Dans tous les cas les informations de dépendance application/interface permettent de connaître les applications concernées par ces suppressions.

6.1.2.3 Documentation découpe produit

Diverses fonctions de documentation sont offertes à l'utilisateur :

- Documentation globale : cette fonction fournit la découpe globale d'un produit en applications, en donnant pour chaque application la liste des interfaces publiques et privées associées, ainsi que les interfaces

privées et publiques utilisées.

- Liste des applications du produit

- Liste des interfaces du produit, classées suivant le critère publique/privée.

- Liste des interfaces d'une application du produit, classées suivant le critère publique/privée.

- Liens : l'utilisateur peut obtenir les liens

  - application / interfaces utilisées

  - interface / applications utilisatrices.

6.1.3 Découpe au niveau application :

Rappel : une application s'exprime en termes de modules.

Les fonctions réalisées par l'éditeur de découpe sont les suivantes :

- initialisation de la découpe.

- modification de la découpe.

- documentation.

6.1.4 Initialisation de la découpe

Cette opération permet de donner une première découpe de l'application. Elle consiste à :

- déclarer les modules en donnant pour chacun, son nom, sa nature (définition, réalisation, interface) et ses directives import et interface.

Les déclarations des modules doivent respecter l'ordre des directives Import,c'est à dire que si un module M2 importe un module M1 alors le module M1 doit être déclaré auparavant. Une erreur est signalée à l'utilisateur pour chaque module définition importé non encore déclaré,cependant le module est crée avec les déclarations Import valides. L'utilisateur doit passer par la fonction modification pour corriger les erreurs. On est ainsi certain de ne pas créer de cycle.

Pour les interfaces un message d'erreur est signalé pour chaque interface inexistante ,c'est à dire ne figurant ni au niveau produit où est développée l'application, ni au niveau projet. Là encore l'utilisateur doit utiliser la fonction modification de l'éditeur de découpe pour corriger les erreurs éventuelles.

La déclaration d'un lien interface provoque :

- la mise à jour de l'objet lien du niveau produit en créant le lien entre l'application courante et l'interface.

- s'il s'agit d'une interface publique la mise à jour de l'objet lien du niveau projet en créant un lien entre le produit , dont fait parti le module, et l'interface .

Les déclarations des modules sont mémorisés dans l'objet liste de composition et les directives Import et Interface dans l'objet lien, niveau

application.

A partir de cette liste de composition on initialise les répertoires des bibliothèques figurant au niveau application (cf. Annexe B Structure d'une bibliothèque).

6.1.4.1 Modification de la découpe

Les opérations de modification sont les suivantes :

- ajout d'un nouveau module en donnant son nom et les directives Import, Interface.

- suppression d'un module.

- modification des directives Import ou Interface d'un module.

Ces opérations de modification mettent à jour la liste de composition de l'application et l'objet lien niveau application.

La suppression d'un module interface n'est possible que si celui-ci a été supprimé au niveau produit.

La modification des directives Import et Interface se fait globalement : c'est-à-dire que les anciennes directives sont supprimées et l'utilisateur donne la nouvelle liste des Import et Interface. La modification de la directive interface ou la suppression d'un module peut provoquer une mise à jour éventuelle de l'objet lien du niveau produit et de l'objet lien du niveau projet. En effet considérons par exemple un module M1 de l'application A1 du produit P1. Lors de la création de ce module l'utilisateur a déclaré :

Interface I1, I2 .

Un lien entre I1(interface publique), et I2 (interface privée de P1) avec M1 a été créé dans l'objet lien du niveau application. Un lien entre l'application A1 et les interfaces I1 et I2 a été créé dans l'objet lien du niveau produit. De même un lien entre le produit P1 et l'interface publique I1 a été créé dans l'objet lien du niveau projet. D'autre part M1 est le seul module de l'application A1 à avoir un lien interface avec I1 et I2. L'utilisateur à l'aide des fonctions de modification change la directive interface de M1 en :

Interface I3.

Alors les liens entre l'application A1 et les interfaces I1 et I2 sont supprimés dans l'objet lien du niveau produit. Il n'y alors plus de lien entre l'application A1 et l'interface publique I1, le lien entre le produit P1 et l'interface publique I1 est supprimé au niveau projet.

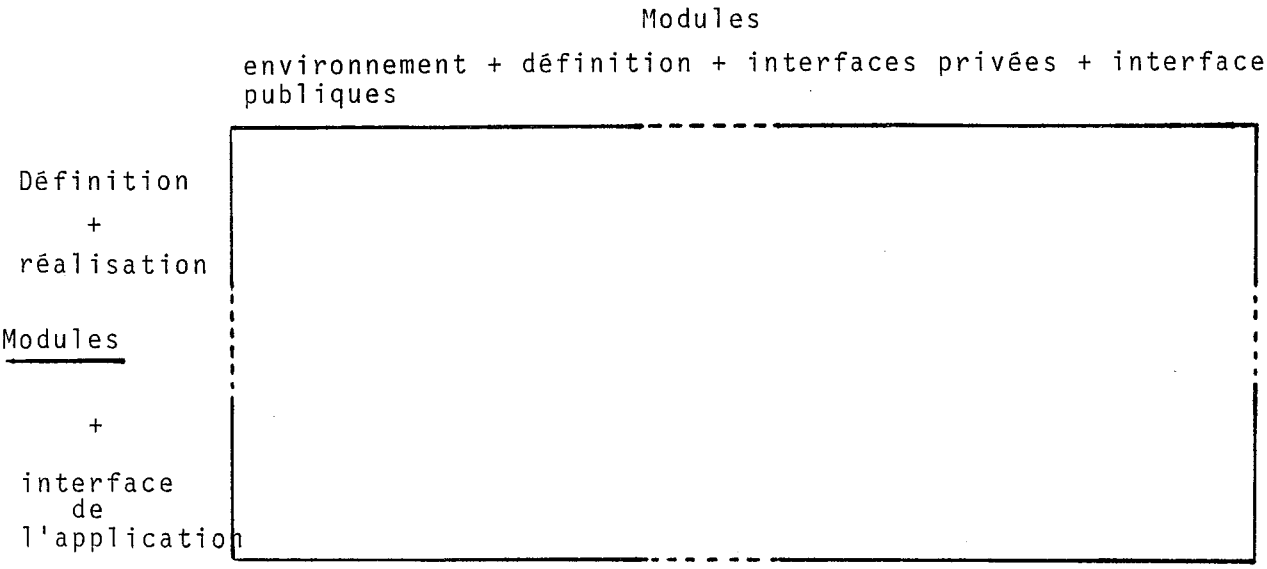
L'utilisateur doit faire attention de ne pas créer de cycle entre les modules définition lors de la modification de la liste des Import d'un module. La création éventuelle d'un cycle n'est pas contrôlée lors de cette opération. L'utilisateur dispose d'une fonction de détection de cycle dans la fonction documentation de l'éditeur de découpe niveau application.

Lors de la suppression d'un module définition ou la modification des directives Import/Interface d'un module définition les liens de dépendance module/module définition permettent de connaître les modules concernés.

6.1.4.2 Documentation découpe application

Les fonctions suivantes de documentation sont offertes :

- documentation globale : cette fonction fournit la découpe de l'application en modules en donnant pour chaque module les modules importés et interfacés. Les modules sont classés par nature : environnement - définition - réalisation - interfaces construites sur l'application. Par ailleurs cette fonction édite la liste de tous les liens inter-modules sous la forme matricielle :



- liste des modules de l'application

- liste des interfaces créées sur l'application

- liens : l'utilisateur peut obtenir les liens :

  - module / modules Importés et Interfacés
  - module définition / modules utilisateurs
  - interface / modules utilisateurs
  - liste des modules à compiler si modification d'un module définition
  - liste des modules à compiler si modification d'un module interface.

- détection de cycle : un cycle peut apparaître entre les modules définition d'une application lors de l'appel de la fonction modification de la directive Import d'un module définition. Détecter un cycle entre modules définition est une opération coûteuse,c'est pourquoi une fonction particulière de l'éditeur de découpe permet de vérifier la présence éventuelle de cycles. L'utilisation de cette fonction est laissée à la discrétion de l'utilisateur.

6.1.5 Les dialogues de la session conception

L'interface usager se fait par un mécanisme de menus hiérarchisés. L'utilisateur sélecte ainsi la fonction qu'il désire réaliser. Il obtient alors soit l'affichage d'un nouveau menu ,soit le dialogue correspondant à la fonction demandée. Les schémas suivants montrent :

- pour le premier l'affichage d'un menu proposant les fonctions de documentation sur la découpe d'un produit.

- pour le second le dialogue correspondant à la création d'un module lors de la phase modification de la découpe d'une application.

ATELIER LOGICIEL INTEGRE S.E.M.S 25 nov 1983

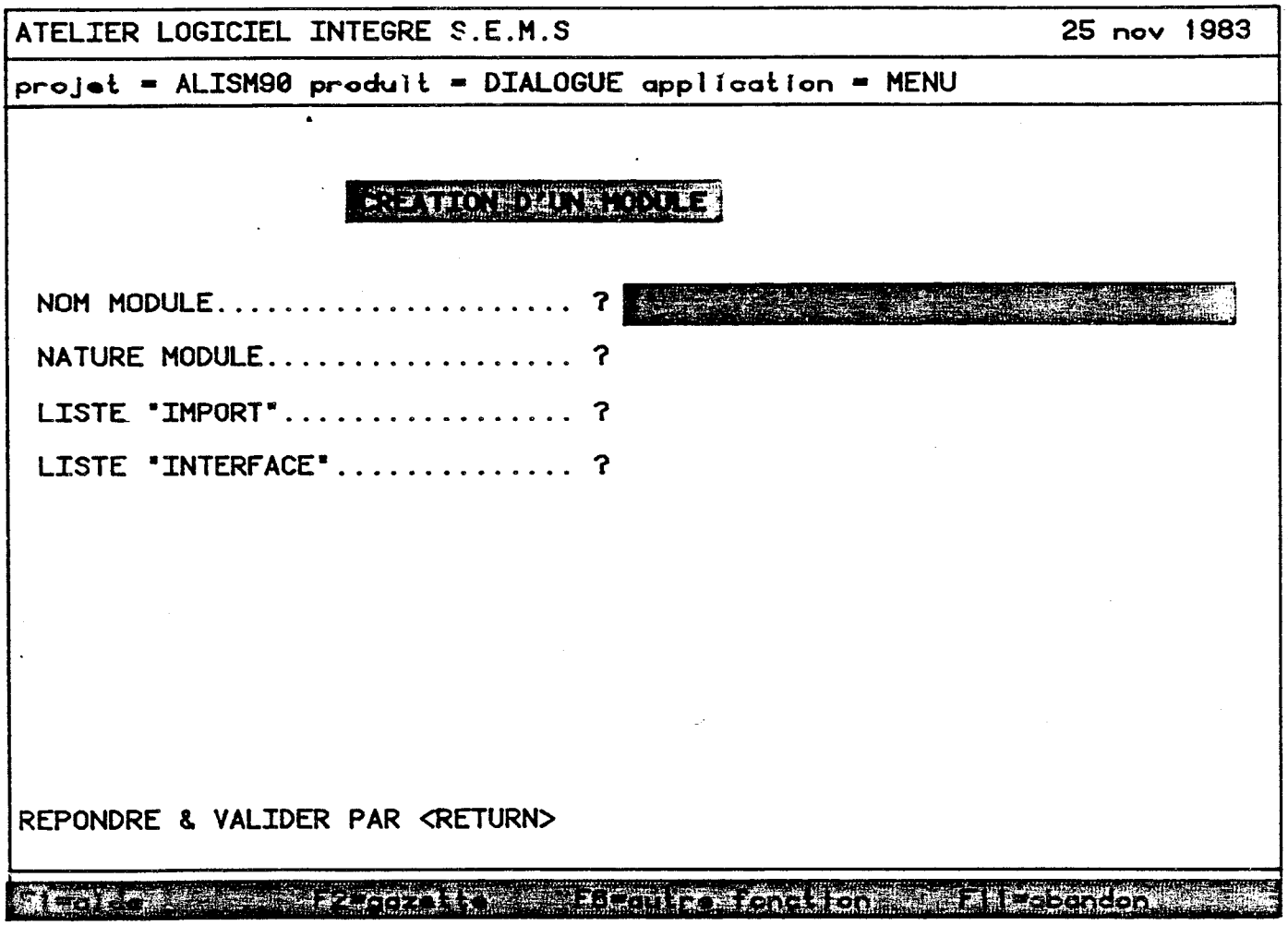
projet = ALISM00 produit = DIALOGUE

[illegible]

- 1 - DOCUMENTATION COMPLETE
- 2 - LISTE DES APPLICATIONS DU PRODUIT
- 3 - LISTE DES INTERFACES APPARTENANT AU PRODUIT
- 4 - LISTE DES INTERFACES D'UNE APPLICATION
- 5 - LISTE DES APPLICATIONS UTILISANT UNE INTERFACE
- 6 - LISTE DES INTERFACES UTILISEES PAR UNE APPLICATION

SELECTER & VALIDER PAR <RETURN>

F1=aide F2=gazette F8=autre fonction F11=abandon


ATELIER LOGICIEL INTEGRE S.E.M.S
25 nov 1983
projet = ALISM90 produit = DIALOGUE application = MENU
CREATION D'UN MODULE
NOM MODULE..... ?
NATURE MODULE..... ?
LISTE "IMPORT"..... ?
LISTE "INTERFACE"..... ?
REPONDRE & VALIDER PAR <RETURN>
F1 Abandon

6.2 OBJETS LIEN ET LISTE DE COMPOSITION

On vient de voir que l'éditeur de découpe travaille essentiellement avec deux objets mémorisant cette découpe : un objet liste de composition et un objet lien. Ces deux objets sont répartis suivant les niveaux projet - produit - application. Les chapitres suivants décrivent en détail les deux objets.

6.2.1 Objet liste de composition

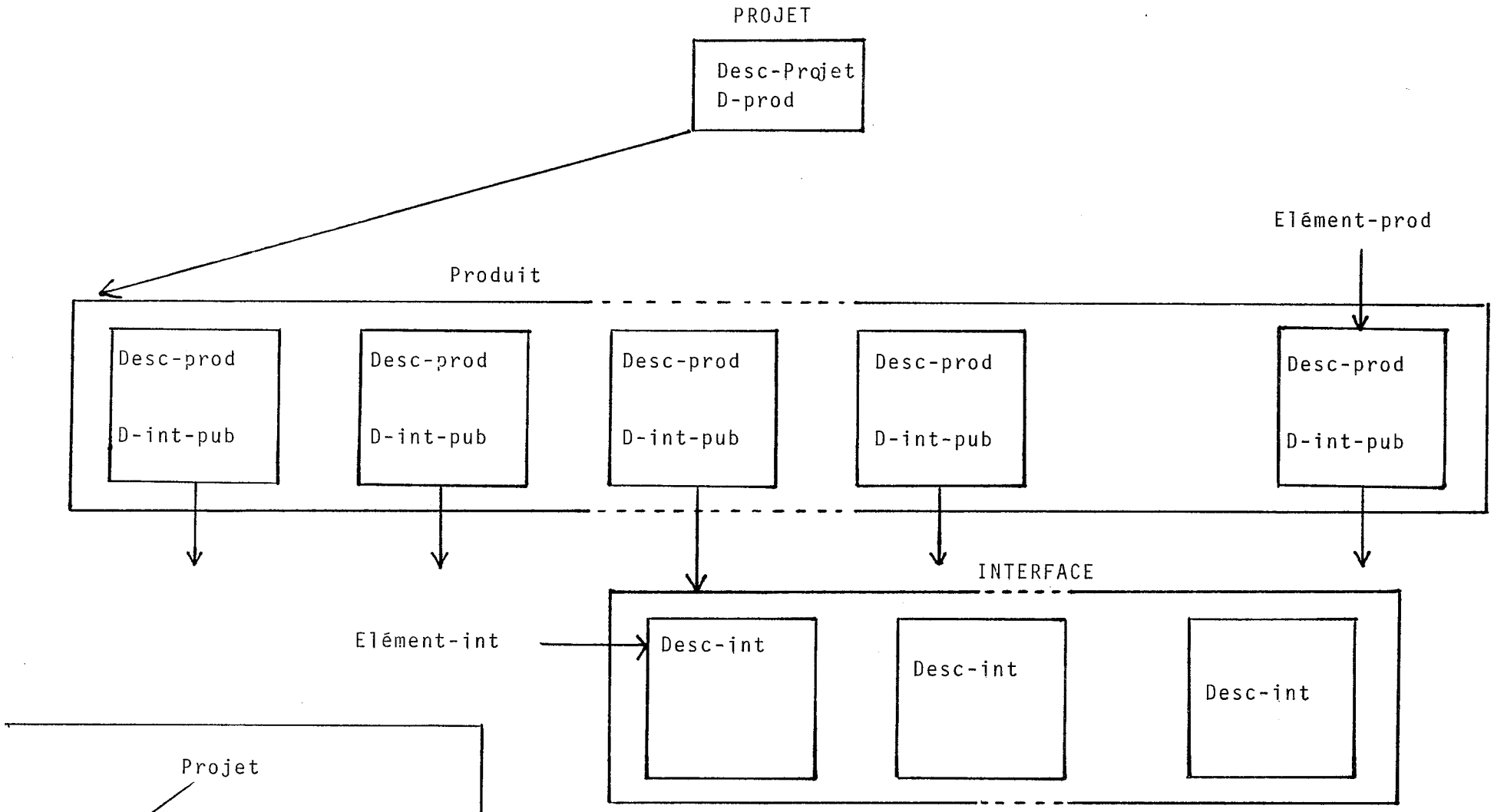
6.2.1.1 Niveau projet

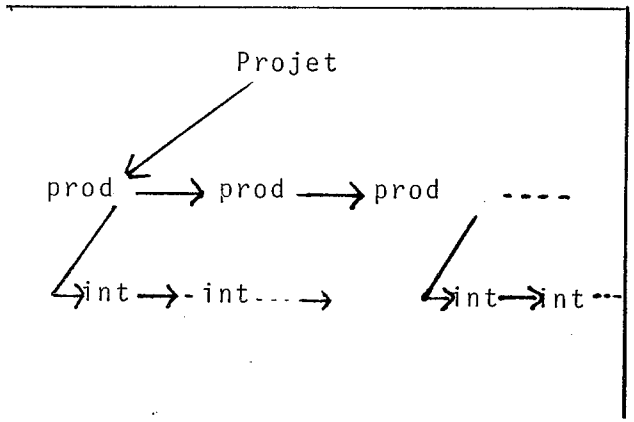
L'objet liste de composition niveau projet contient les informations relatives aux différents produits : il contient un article par produit. Il est initialisé et mis à jour par l'éditeur de découpe.

La structure de l'objet liste de composition est arborescente sur deux niveaux :

- 1er niveau : chaque article correspond à un produit
- 2ème niveau : pour chaque produit on trouve ses interfaces publiques.

L'accès à cet objet se fait pour un jeu de primitives décrites dans l'interface I : PRO (cf. Annexe C).





OBJET Liste DE Composition
niveau PROJET

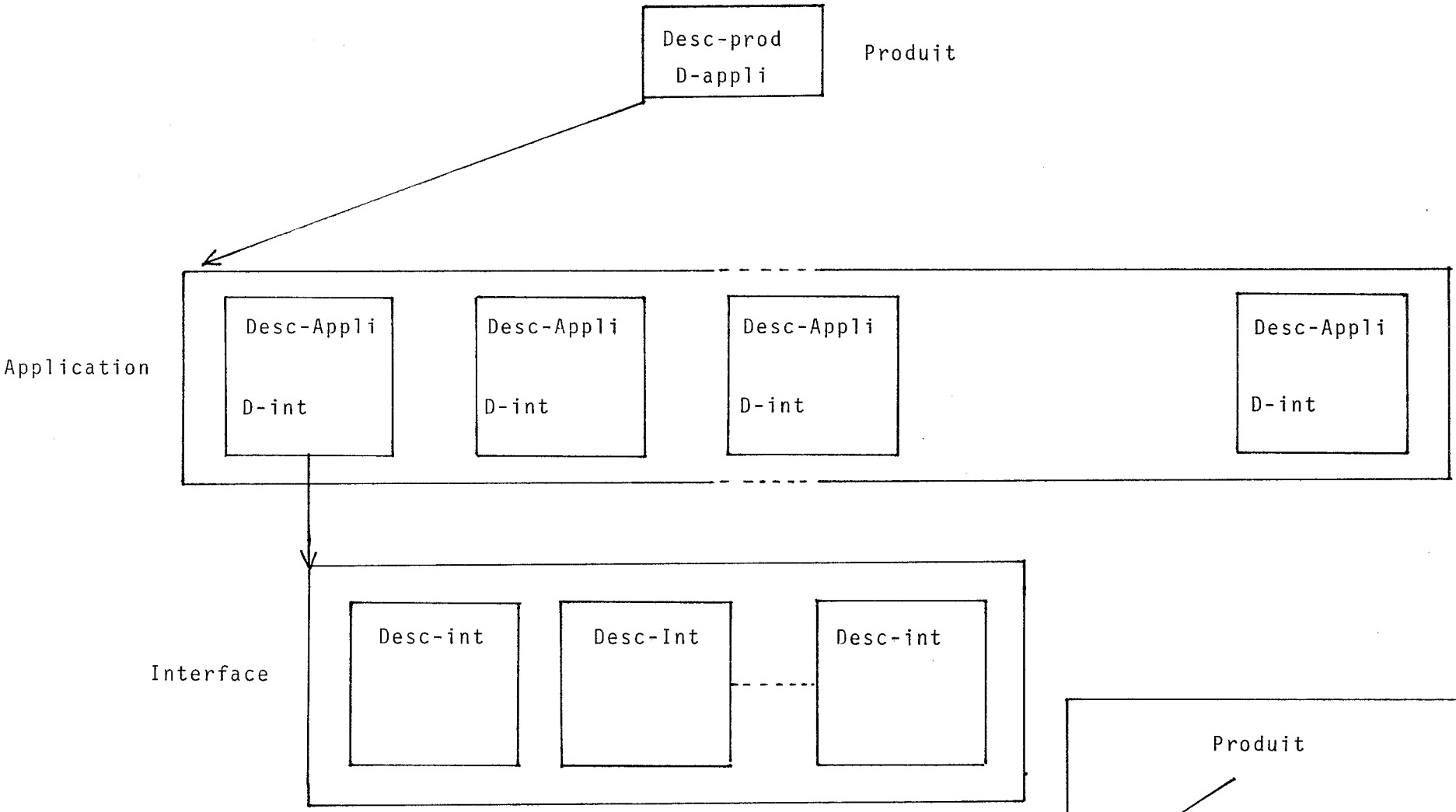
6.2.1.2 Niveau produit

L'objet liste de composition niveau produit contient les informations relatives aux différentes applications du produit. Il est initialisé et mis à jour par l'éditeur de découpe.

Sa struture est arborescente sur deux niveaux :

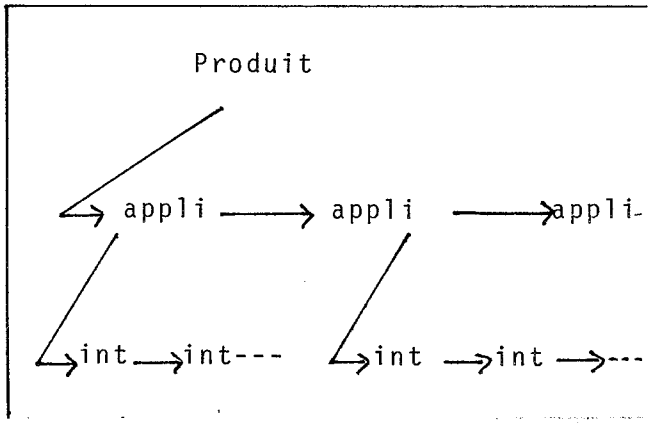
- 1er niveau : chaque article décrit une application du produit
- 2ème niveau : pour chaque application on trouve la liste des interfaces associés.

L'accès à cet objet se fait par l'intermédiaire d'un jeu de procédures décrites dans l'interface I : PRD (cf. Annexe C).



OBJET Liste de COMPOSITION

niveau PRODUIT



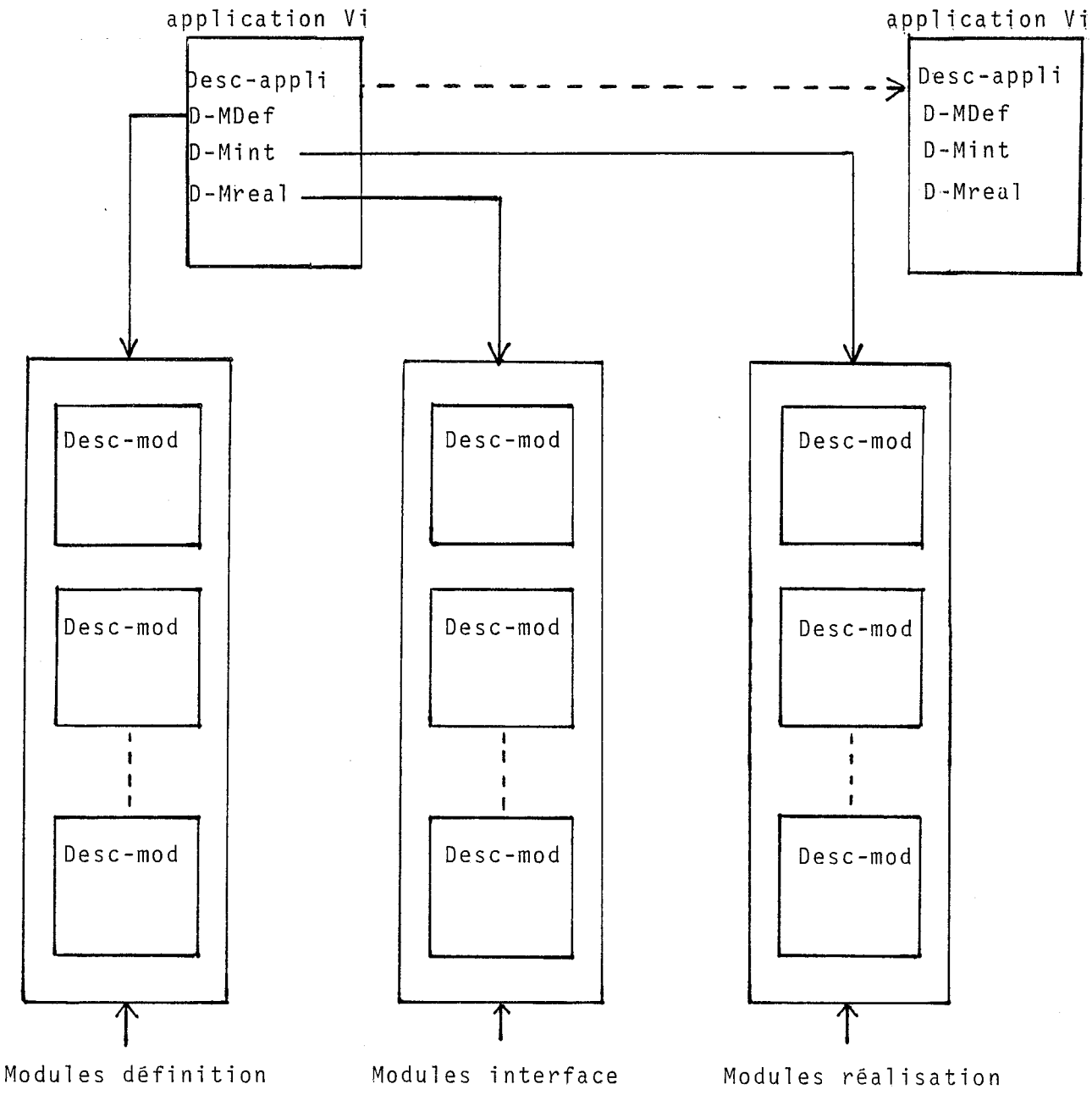
6.2.1.3 Niveau Application

L'objet liste de composition niveau application contient les informations relatives aux différents modules de l'application. Il est initialisé et mis à jour par l'éditeur de découpe.

Cet objet ne contient pas les liens import et interface (cf. Chapitre modularité dans Alis), ces liens sont mémorisés dans un objet lien décrit au chapitre suivant.

La structure de cet objet est simple. Chaque article correspond à un module, les modules de même nature (i.e. Définition, réalisation, interface, environnement) étant regroupés.

L'accès à cet objet est possible par l'intermédiaire d'un jeu de procédures décrites dans l'interface I : APL (cf. Annexe C).



OBJET Liste de Composition
niveau application

6.2.2 Objet liens

La gestion des liens est répartie suivant les trois niveaux constituant l'architecture de l'atelier ALIS : Projet / produit / application.

6.2.2.1 Niveau projet :

On gère à ce niveau les interfaces publiques à un projet. On mémorise les liens suivants :

Produit <--> interfaces publiques c'est-à-dire :

Quel(s) produit(s) utilise(nt) l'interface publique XX .

Quelle(s) est(sont) le(s) interface(s) publique(s) utilisée(s) par le produit YY.

6.2.2.2 niveau produit

Un produit est un ensemble d'applications. On mémorise à ce niveau les liens applications <--> interfaces privées et publiques c'est-à-dire :

Quelle(s) application(s) utilise(nt) l'interface publique ou privée XX.

Quelle(s) est(sont) le(s) interface(s) utilisée(s) par l'application YY.

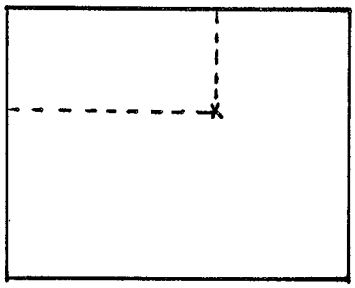
6.2.2.3 niveau application

Une application est un ensemble de modules

On mémorise à ce niveau les liens entre les modules définition + réalisation + interface de l'appli <--> modules environnement + définition de l'application + interfaces publiques

On mémorise donc directement les directives Import et Interface figurant dans la déclaration de chaque module.

Produits

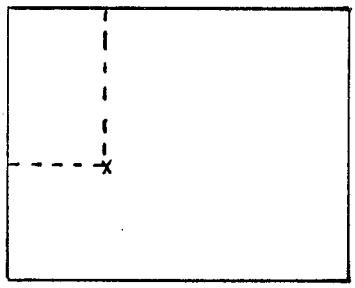


Niveau projet

Interfaces publiques

Niveau sous-projet

Applications



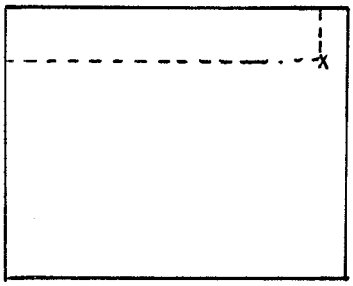
Interfaces publiques

+

privées

Niveau application

Modules environnement+définition+interfaces

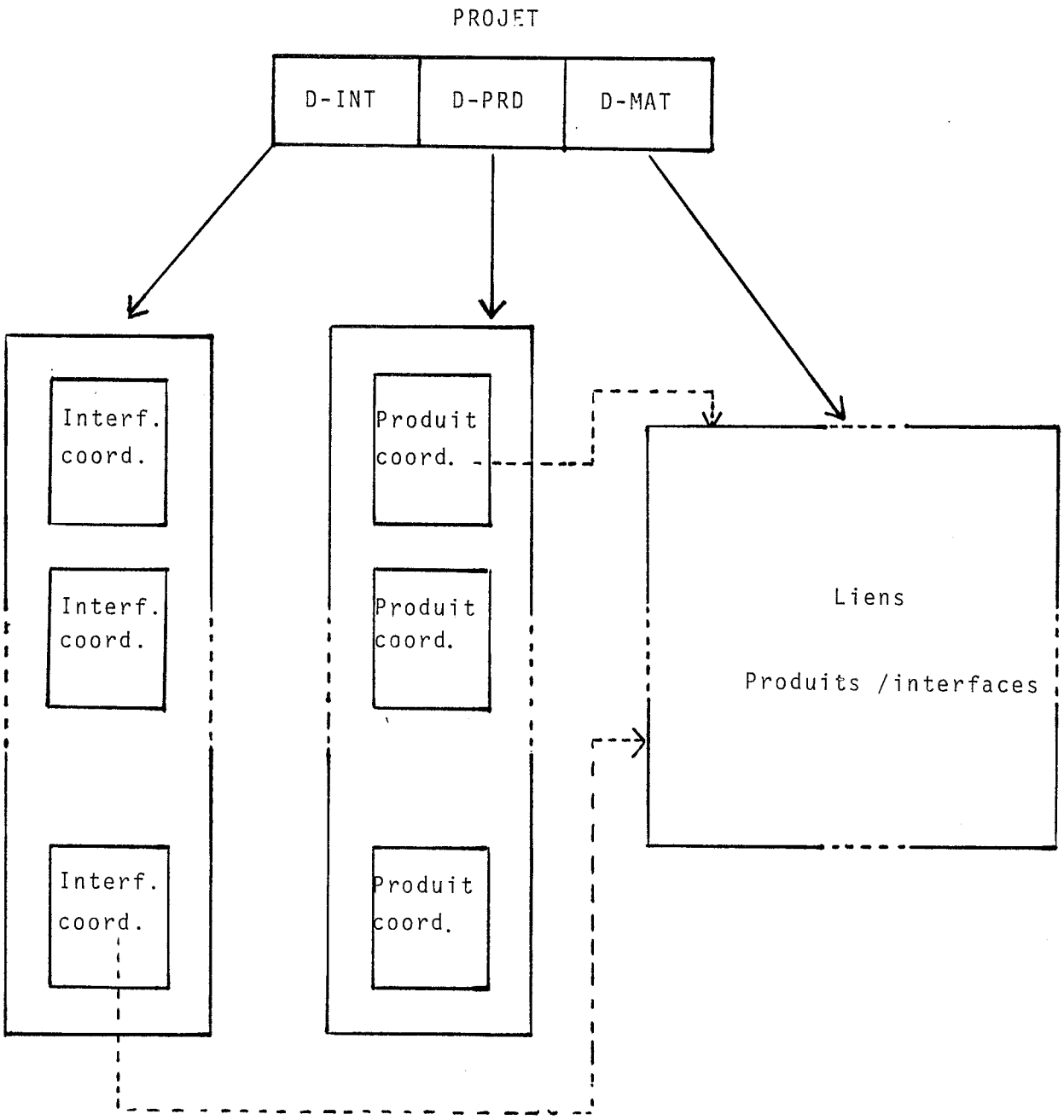


Modules Définition(s) + réalisation(s) +(interface(s) de l'appli)

OBJET Lien - REPARTITION

6.2.2.4 Structure interne de l'objet lien

1 - Au niveau projet



OBJET Lien - Niveau Projet

L'objet lien niveau projet contient :

- une matrice booléenne exprimant les liens produit <--> interfaces publiques

- Un "répertoire" des interfaces publiques permettant de connaître le numéro de ligne de chaque interface.

- Un "répertoire" des produits permettant de connaite le numéro de colonne de chaque produit.

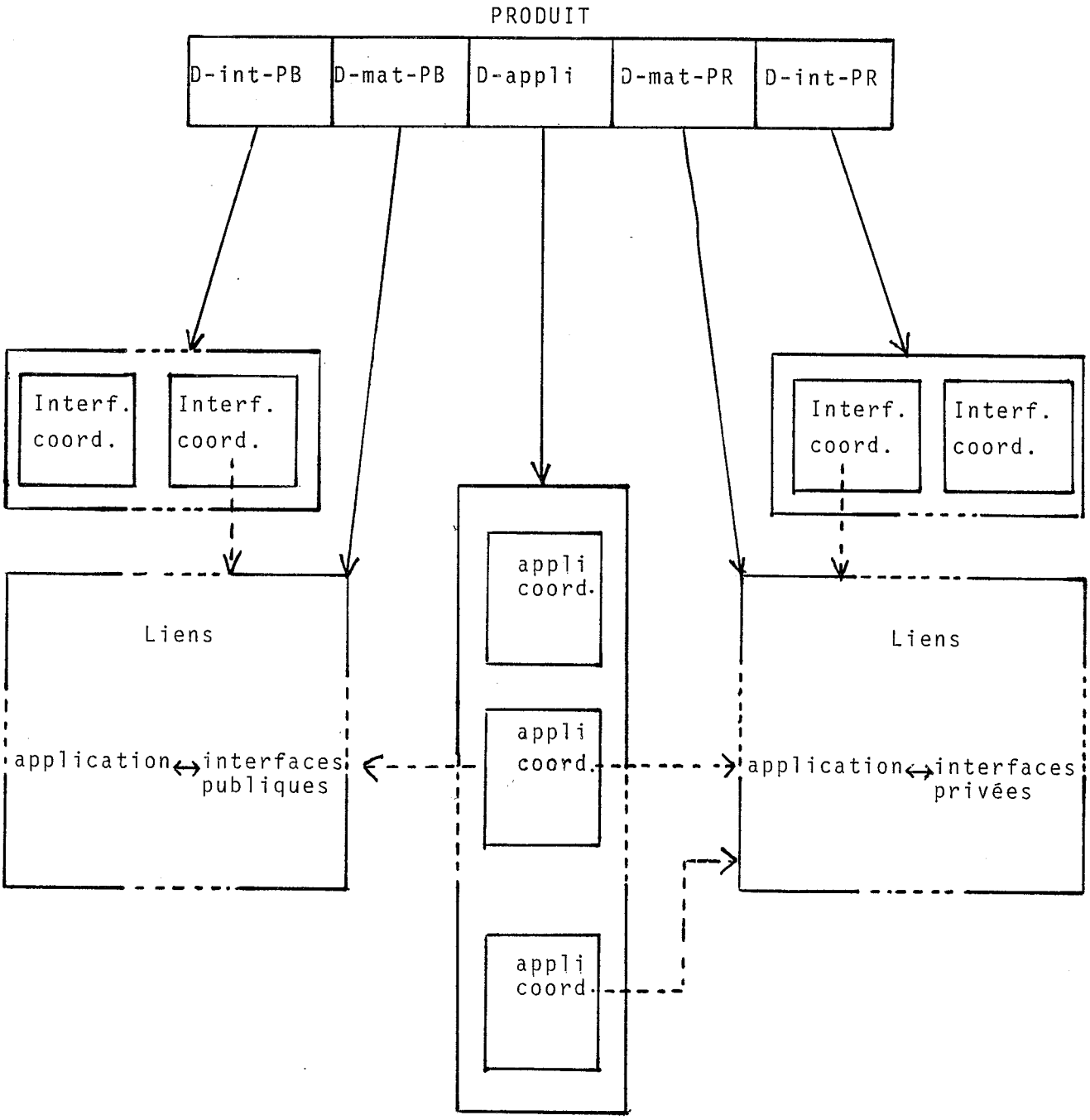
Les répertoires sont créés :

- Lors de la phase conception de projet à partir de la liste de composition du projet. Ils sont mis à jour lors d'éventuelles modifications de cette liste.

Les liens sont mis à jour :

- lors de la phase conception de l'application à partir de la déclaration des liens interfaces des modules des application du projet ou de la modification des liens d'un module.

2 - au niveau produit



OBJET Lien - niveau PRODUIT

L'objet lien niveau produit contient :

- une matrice booléenne exprimant les liens entre les applications du produit et les interfaces publiques.

- une matrice booléenne exprimant les liens entre les applications du produit et les interfaces privées.

- un répertoire des applications du produit permettant de connaître les numéros de ligne de chaque application dans les deux matrices.

- un répertoire des interfaces publiques utilisées permettant de connaître le numéro de colonne de chaque interface dans la matrice booléenne correspondante.

- un répertoire des interfaces privées permettant de connaître le numéro de colonne de chaque interface privée dans la matrice booléenne correspondante.

Ces différents répertoires sont créés:

- pour le répertoire des applications : lors de la phase conception du produit à partir de la liste de composition du produit en termes d'application. Il est ensuite mis à jour lors d'éventuelles modifications de cette liste.

- pour les répertoires des interfaces privées lors de la phase conception du produit à partir de sa liste de composition (pour chaque application on précise les interfaces privées associées). Ce répertoire est mis à jour lors d'éventuelles modifications de cette liste.

- pour le répertoire des interfaces publiques, il est initialisé à vide lors de la phase conception du produit, et mis à jour éventuellement à partir des déclarations des liens interfaces des modules des différentes applications du produit.

3 - au niveau application

La gestion des liens de dépendance s'appuie sur un objet lien contenant une matrice (n, m) de booléen représentant les liens inter-modules.

Elle permet de répondre aux questions :

Quel(s) module(s) définition importe(nt) le module xx

Quel(s) sont le(s) module(s) important le module définition yy

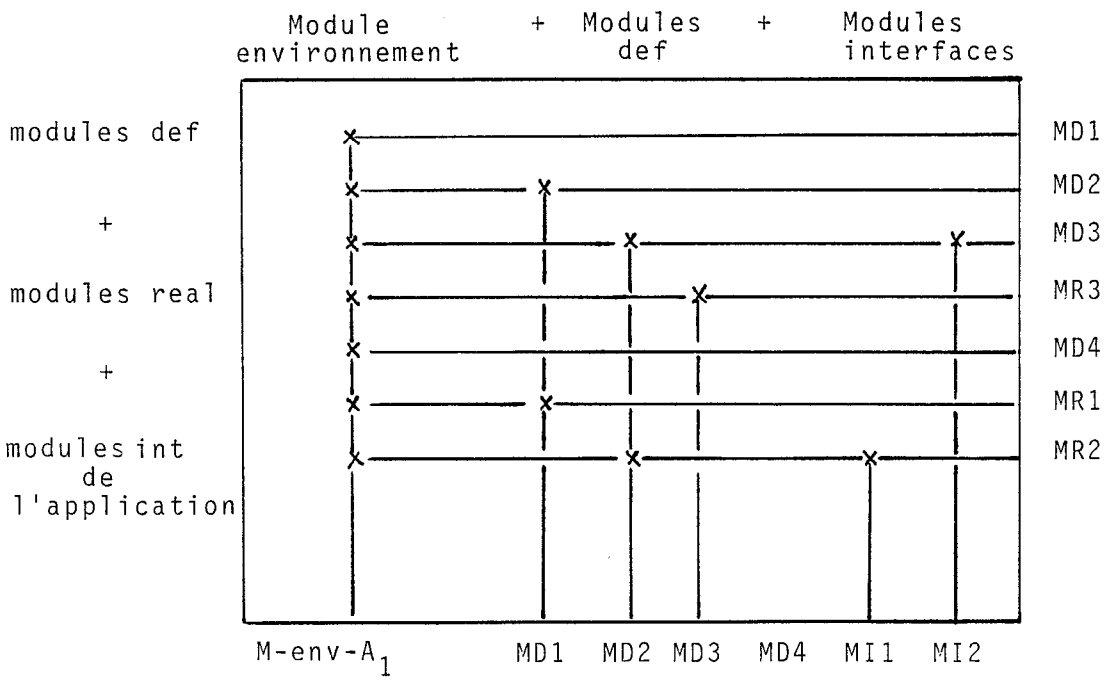
Quel(s) module(s) interface importe le module xx

Quel(s) sont le(s) module(s) important le module interface yy.

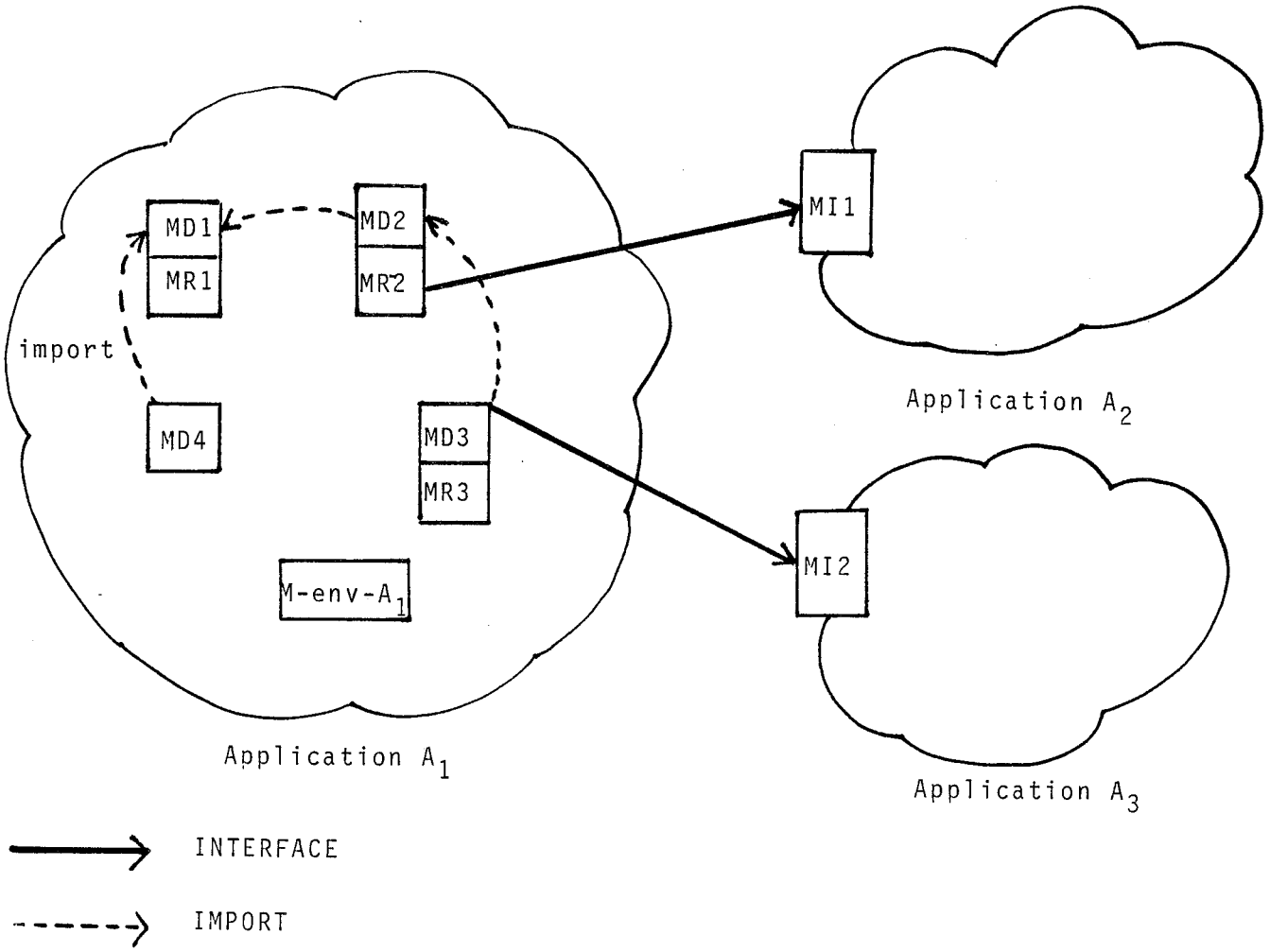
Pour tenir compte du problème de gestion des versions et pour tenir compte des solutions choisies dans la gestion des bibliothèques (système de répertoires-objets) l'objet lien est organisé, pour chaque version d'une application, en deux parties :

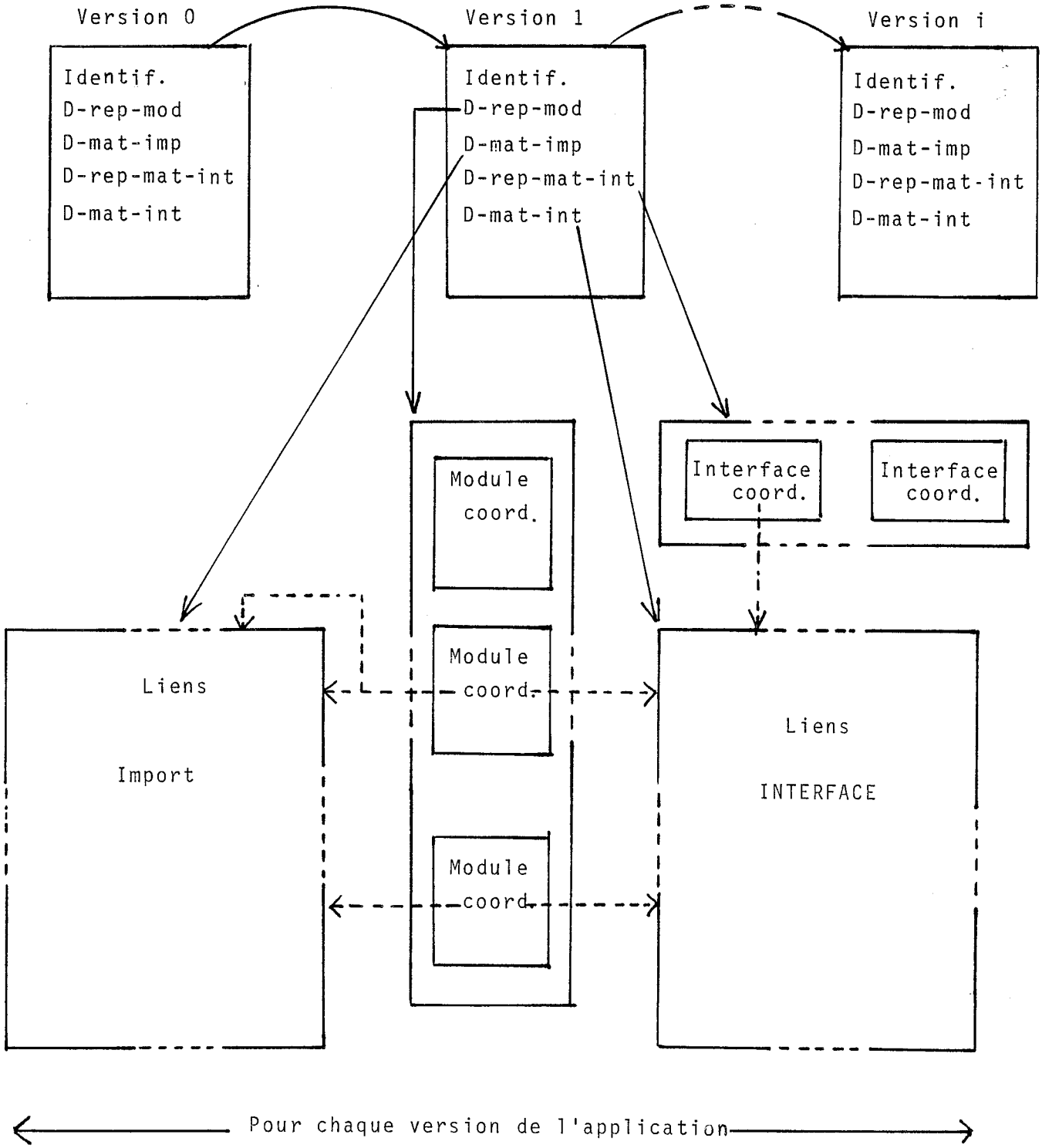
- une partie répertoire qui contient l'identification de chaque module et ses coordonnées dans la matrice.
- la matrice elle-même : qui est dynamique dans sa taille. Cette matrice n'est pas carrée : un module réalisation peut importer des modules définition, mais on ne peut importer un module réalisation.

Exemple de matrice



Cette matrice correspond à la découpe suivante :





OBJET Lien - Niveau application

Les répertoires sont créés :

- lors de la phase de conception de l'application à partir de la liste de composition en termes de modules. Les liens aussi sont créés à partir de cette liste.

- le répertoire et les liens sont mis à jour lors d'éventuelles modifications de cette liste.

Les interfaces gérant les divers objets liens sont décrits en annexe C.

7 UTILISATION DES INFORMATIONS GEREES PAR LA SESSION CONCEPTION

Toute la découpe d'un logiciel décrite par l'utilisateur d'Alis étant mémorisée, les outils des différentes sessions d'Alis s'appuient directement sur les informations de découpe.

7.1 LA SESSION PRODUCTION DE PROGRAMMES

7.1.1 Présentation générale

L'organisation de la chaîne de production de programme est classique. Elle comprend dans ses grandes lignes :

- des compilateurs.
- un générateur-optimiseur par machine cible.
- un éditeur de liens.
- un générateur d'images mémoire.
- un debugger symbolique.

7.1.1.1 Compilateur et générateur-optimiseur

Le concept de modularité a conduit à écrire un compilateur permettant de "compiler dans un environnement". Habituellement les compilateurs réalisent un contrôle strict de l'emploi des types, des variables et des paramètres de procédure ou fontion au niveau de l'unité de compilation. Cette situation laisse trop de liberté, permet de contourner les contrôles et est la source inévitable des problèmes futurs. Dans Alis les directives Import et Interface, associées à chaque module, définissent le contexte de compilation du module. On étend ainsi au niveau du système complet les contrôles faits habituellement sur l'unité de compilation [COQ 81] .

Pour atteindre cet objectif le processus de compilation est divisé en deux parties :

- la gestion de l'environnement de compilation : on conserve les tables des symboles des modules définition et interface dans des bibliothèques
- la compilation traditionnelle qui traduit la partie réalisation d'un module en effectuant tous les contrôles syntaxiques et sémantiques grâce aux tables des symboles de tous les modules visibles gardées dans le contexte de compilation.

Ce processus de compilation séparée ne peut fonctionner qu'en respectant un ordre de compilation des modules :

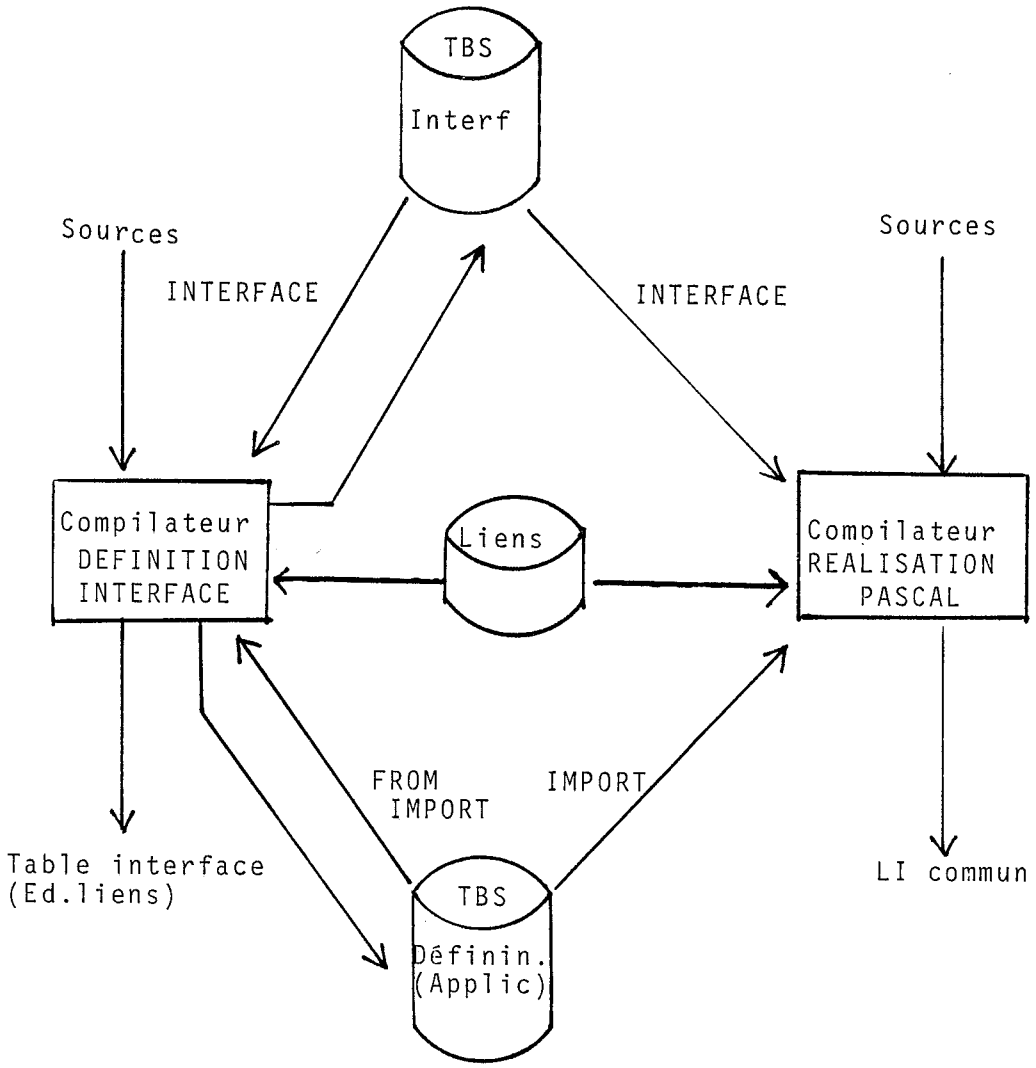
- compiler les modules en suivant l'ordre des liens de visibilité exprimés par les liens Import et Interface. Il faut noter que l'on peut écrire tous les modules définition et interface sans

écrire les modules réalisation.

- ne pas avoir de cycle entre les modules définitions.

Par ailleurs le résultat d'une compilation d'un module réalisation est exprimé dans un langage intermédiaire LI. Ce LI est commun à tous les compilateurs ce qui permet d'avoir un seul générateur (par machine cible), qui accepte donc un programme en format LI et le traduit en binaire translatable Alis (BT-Alis). Ce générateur étant unique, pour une machine cible donnée, un effort important a été fait pour produire un bon code objet. Il comporte des traitements particuliers pour la gestion des registres et l'optimisation des branchements, et réalise deux optimisations :

L'une dite globale optimise le programme LI en faisant des mises en facteur de sous-expressions communes, des optimisations de calcul d'index pour les tableaux, le calcul des expressions de constantes ou l'élimination des instructions inéffectives. L'autre dite locale utilise la technique nommée "peephole" et s'intéresse à la réduction systématique de certaines séquences du code objet produit, pour utiliser au mieux le jeu d'instructions de la machine cible.



Environnement de Compilation

7.1.1.2 Editeur de liens et générateur d'images mémoires

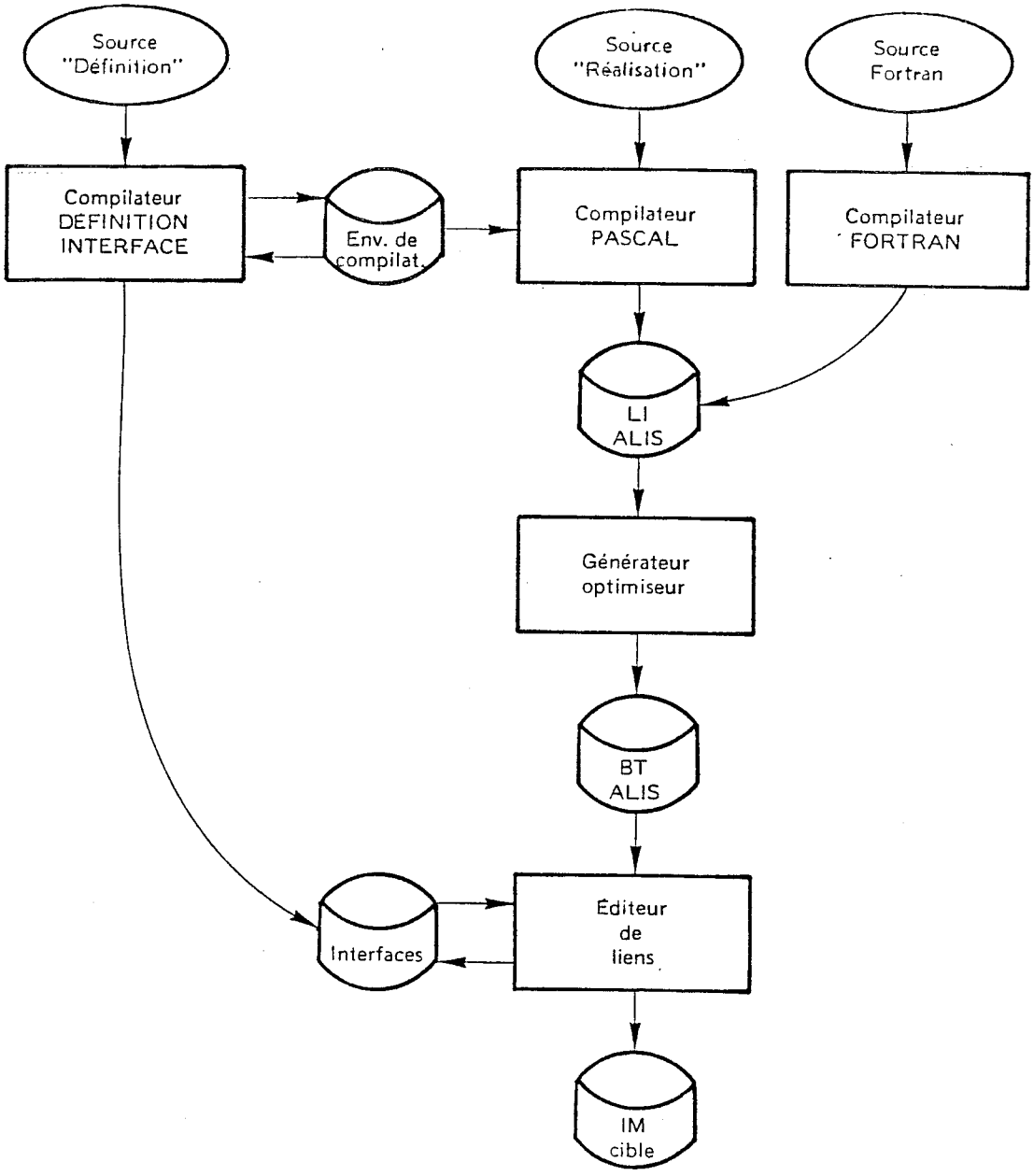
L'éditeur de liens comprend deux fonctions principales [EDL 81] :

- le lieur qui rassemble les BT-Alis d'une même application et résoud les liens de type import entre ces modules.

- le connecteur qui résoud les liens de type interface avec les autres applications.

Le résultat de l'éditeur de liens est une image mémoire translatable (IMT) et une table d'interface contenant les noms et adresses des procédures de l'interface.

Le générateur d'images mémoires produit à partir des IMT des images mémoires exécutables au format de la machine cible.



Organisation de la production de programmes dans ALIS

7.1.2 Création du contexte de compilation d'un module

Le contexte de compilation d'un module comprend :

- Les tables des symboles des modules interfaces dont les noms sont déclarés dans les directives interface.

- Les tables des symboles des modules définition dont les noms sont déclarés dans les directives import.

- Le module environnement de l'application à laquelle appartient le module. (Le module environnement est un module définition particulier qui est "importé" automatiquement par tout module de l'application).

La phase de création de ce contexte consiste à parcourir l'objet lien niveau application, qui mémorise toutes les directives Import et Interface des modules de l'application, et à rechercher les tables des symboles des modules "importés" et "interfacés". Si une de ces tables n'existe pas, ou est invalide, c'est-à-dire que l'ordre de compilation n'a pas été respecté, l'utilisateur en est informé et la compilation n'a pas lieu.

7.1.3 "Re"-compilations automatiques et re-édition des liens

Les directives Import et Interface définissent un ordre de compilation (cf. Chapite modularité dans Alis). Cette dépendance entre les modules est conservée par le graphe de dépendance mémorisé dans l'objet lien. Ce graphe est exploité dans plusieurs cas :

7.1.3.1 Compilation d'un module définition

Le compilateur lors de la compilation d'un module définition, donne la liste des modules à recompiler pour conserver la cohérence de compilation. Si l'utilisateur le désire les "re"-compilations des modules concernés peuvent être lancés automatiquement.

7.1.3.2 livraison d'une interface

Rappel : une interface est, dans une première phase, créée dans l'application mère. Elle s'exprime par :

- une table des symboles TBS

- un fichier d'interface TBI

- une image mémoire IMT.

Dans une deuxième phase, cette interface est "livrée" au niveau produit, pour devenir utilisable par toutes les applications du produit. La livraison consiste donc à transférer du niveau application au niveau produit les 3 éléments TBS, TBI, IMT (ou une combinaison de ces 3 éléments). Lors de la livraison le graphe de dépendance figurant au niveau produit et conservé dans l'objet lien permet de connaître toutes les applications utilisatrices de l'interface transférée. Ensuite au niveau de chaque application utilisatrice, le graphe de dépendance entre les modules permet de connaître précisément les modules concernés par la livraison.

Dans une troisième phase, si l'interface est publique au projet, il faut la livrer au niveau projet, pour devenir utilisable par tous les produits du projet. Le graphe de dépendance figurant au niveau projet permet de connaître tous les produits utilisateurs de l'interface, puis au niveau de chaque produit utilisateur le graphe de dépendance permet de connaître les applications utilisatrices, puis au niveau des applications utilisatrices le graphe de dépendance inter-modules permet de connaître les modules à

recompiler.

Dans le cas où la livraison consiste à livrer seulement l'élément IMT, c'est-à-dire que seule la réalisation de l'interface a changé, les graphes de dépendance permettent de déterminer pour quelles applications il faut refaire l'édition des liens.

7.1.4 Génération des commandes à l'éditeur des liens

L'éditeur de liens (cf. Chapitre Session Production Programme) comprend une phase dite connecteur qui résoud les liens de type interface entre les applications dont on fait l'édition et les autres applications.

Cette fonction de connection peut être réalisée soit à partir d'un langage de commande dans lequel l'utilisateur spécifie les applications à concaténer, soit à partir de l'objet lien niveau produit qui décrit les liens Application/interfaces utilisés. Ainsi automatiquement l'éditeur de lien obtient les images mémoires des applications à concaténer.

7.1.5 Compleximètre

Le compleximètre est un outil destiné à mesurer la complexité d'un logiciel. Notamment il permet de calculer un degré de modularité du logiciel. Pour cela il utilise directement l'objet lien pour calculer une série de métriques relatives à la modularité :

- au niveau module : en tenant compte des liens figurant dans les directives Import et Interface
- au niveau application : en tenant compte des liens interface avec les autres applications[VER 83].

7.1.6 Documentation

Tout un aspect documentation est offert à l'utilisateur :

- arbre de dépendances entre les modules, entre les applications, entre les produits, ce qui fournit à l'utilisateur un "planning Pert" des opérations qu'il doit effectuer (ordre de compilation des modules, des applications). L'utilisation d'écrans graphiques, dans le cadre d'un poste de travail personnalisé, doit permettre de réellement visualiser la découpe d'un logiciel sous la forme d'un arbre : au niveau projet le squelette général de ce dernier, puis par "zoom" successifs l'utilisateur obtient la visualisation de la découpe d'un produit en applications, et puis celle d'une application en modules.

7.2 EXTENSIONS

7.2.1 Session mise au point - debugger symbolique

Un des objectifs principaux d'Alis est le développement de logiciels exploités sur des machines cibles différentes de la machine atelier. La mise au point implique donc l'exécution en atelier de tous les logiciels développés en atelier. Il faut donc définir une représentation exécutable d'un programme en atelier .

7.2.1.1 Présentation générale du debugger symbolique

L'étape de mise au point est bien souvent l'étape la plus longue dans la phase de développement d'un produit logiciel . Ceci est en grande partie dû à la lenteur et à la lourdeur du cycle : correction - compilation - édition des liens - exécution.

Le debugger symbolique permet le raccourcissement de ce cycle [DBG 83] . Il travaille pour cela sur une représentation intermédiaire dont le cycle de production est plus court.

Par ailleurs cet outil doit tenir compte de la modularité induite par Alis (cf chapitre Modularité dans Alis) et s'appuie pour cela sur les objets liste de composition et lien du bibliothecaire.

La session de mise au point se passe au niveau d'une application. On distingue trois grandes fonctions :

- gestion des modules
- gestion des unités exécutables
- fonction d'exécution

7.2.1.2 gestion des modules

Cette fonction gère la représentation intermédiaire (RI) du module.

Cette RI comprend :

- la table des symboles du module definition (si celui ci existe)
- les liens (import et interface) mémorisés dans l'objet lien
- la table des symboles pour le module réalisation
- la représentation du module en langage intermédiaire (LI)

7.2.1.3 gestion des unités éxécutables

L'unité d'éxécution (UE) est l'unité de debug. Elle comporte l'ensemble de tous les modules importés directement et indirectement par un module racine . Une UE est donc directement concernée par toute modification des liens. C'est pouquoi l'objet lien niveau application va gérer les UE définies par l'utilisateur.

L'utilisation des informations de lien a lieu :

-lors de la phase de création d'une UE. L'utilisateur désigne le module d'entrée dans l'UE . A l'aide des informations de liens

de l'application ,l'outil de création définit ainsi le graphe de l'UE :

-les noeuds sont les modules definition (import) et interfaces (interface) utilisés.

-les branches sont les liens qui les relient.

L'objet lien mémorise ce graphe.

-lors de la phase conception de l'application une modification de la découpe peut concerner une ou plusieurs UE (de la même façon qu'elle peut concerner d'autres modules). L'utilisateur est ainsi informé de la liste des UE concernées par une opération de modification de la découpe d'une application.

7.2.2 Transport sur SM90

L'atelier logiciel Alis doit être porté sur la machine SM90 sous système UNIX. Dans le cadre de cette opération deux premières extensions sont envisagées pour la partie conception d'Alis :

- génération automatique des fichiers 'makefile' utilisés par l'utilitaire du système UNIX : MAKE [FEL 79]. Make est un outil de construction de programmes . Les informations de dépendances entre les fichiers sources et objets(binaires) et les commandes nécessaires pour contruire un programme sont conservées dans un fichier appelé MAKEFILE. Make s'appuie sur la notion de date associée à chaque fichier et teste le graphe de dépendance contenu dans le fichier Makefile pour reconstruire ce qui est nécessaire. Sous UNIX la création du fichier makefile est à la charge de l'utilisateur. Cette génération du fichier Makefile est envisageable à partir des informations de lien. Ceci permettrait d'assurer la cohérence du logiciel produit dans Alis .

- prise en compte des langages C et assembleur 68000. L'atelier logiciel Alis est actuellement très fortement dédié au langage Pascal/S. Pour prendre en compte les langages C ,très utilisés sous UNIX,et le langage assembleur 68000 des passerelles doivent être posées entre Pascal/s et ces deux langages. Notamment certains modules d'une application pourront être écrits en C ou assembleur. Cela signifie que dans la session conception la déclaration d'un module doit comporter en plus de son nom ,sa nature,le langage de programmation. Dans le cas où ce module est écrit en C ou Assembleur il n'y a pas de déclaration de directives Import ou Interface.

8 CONCLUSION

Une première version d'Alis existe sur le matériel Mitra offrant principalement :

- la session réalisation et tests avec les outils de la chaîne de production de programmes.

- la session communication.

- la session conception : les éditeurs de découpe correspondant aux niveaux projet,produit,application sont écrits. Ils s'appuient sur le bibliothécaire décrit dans ce document.

Par ailleurs tout le logiciel écrit pour le projet Alis, l'a été en simulant Alis. C'est le cas du compilateur Pascal/S qui a été simulé à l'aide de pré-processeurs qui ne manipulent que du source , et en particulier la modularité, point fort du langage, est elle totalement simulée. Cela signifie que tout le logiciel a été écrit suivant la découpe projet-produit-application-module . Une des premières applications des outils de découpe a été de décrire la découpe d'Alis lui-même , ce qui fournit une aide appréciable aux équipes de développement du projet Alis (par exemple l'impact de la modification d'une interface) .

La deuxième version est prévue sur le matériel SM90 et le poste scientifique. Dans ce cadre les dialogues avec l'usager pourront être améliorer (utilisation d'un écran multi-fenêtres), et les extensions décrites au chapitre précédent, concernant la session conception, pourront être implantées.

9 ANNEXES

Annexe A : évolution des coûts du logiciel

Annexe B : structure d'une bibliothèque gérée par bibliothécaire

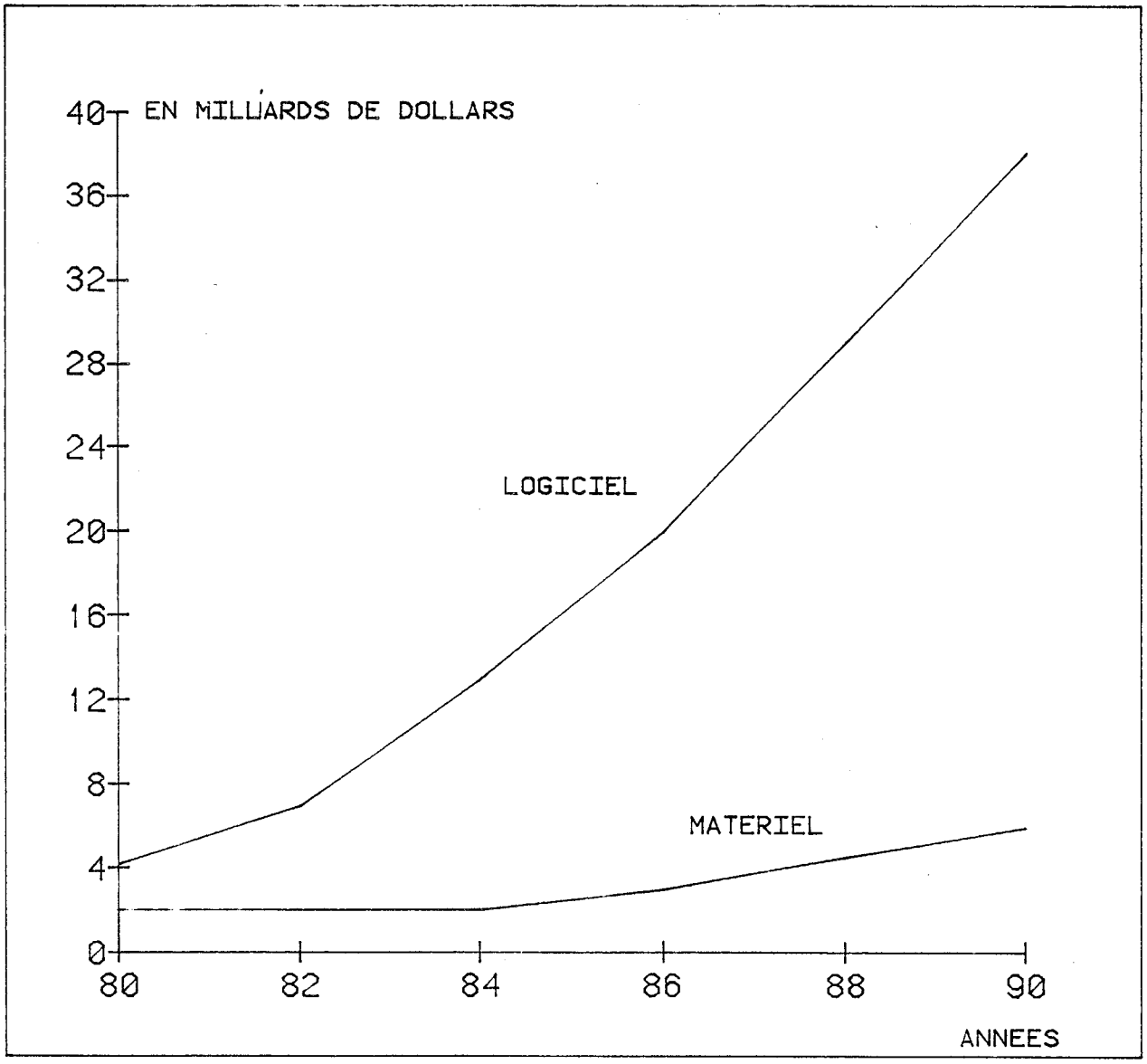
Annexe C : interface objet(s) lien et liste de composition

Annexe E : présentation de quelques ateliers

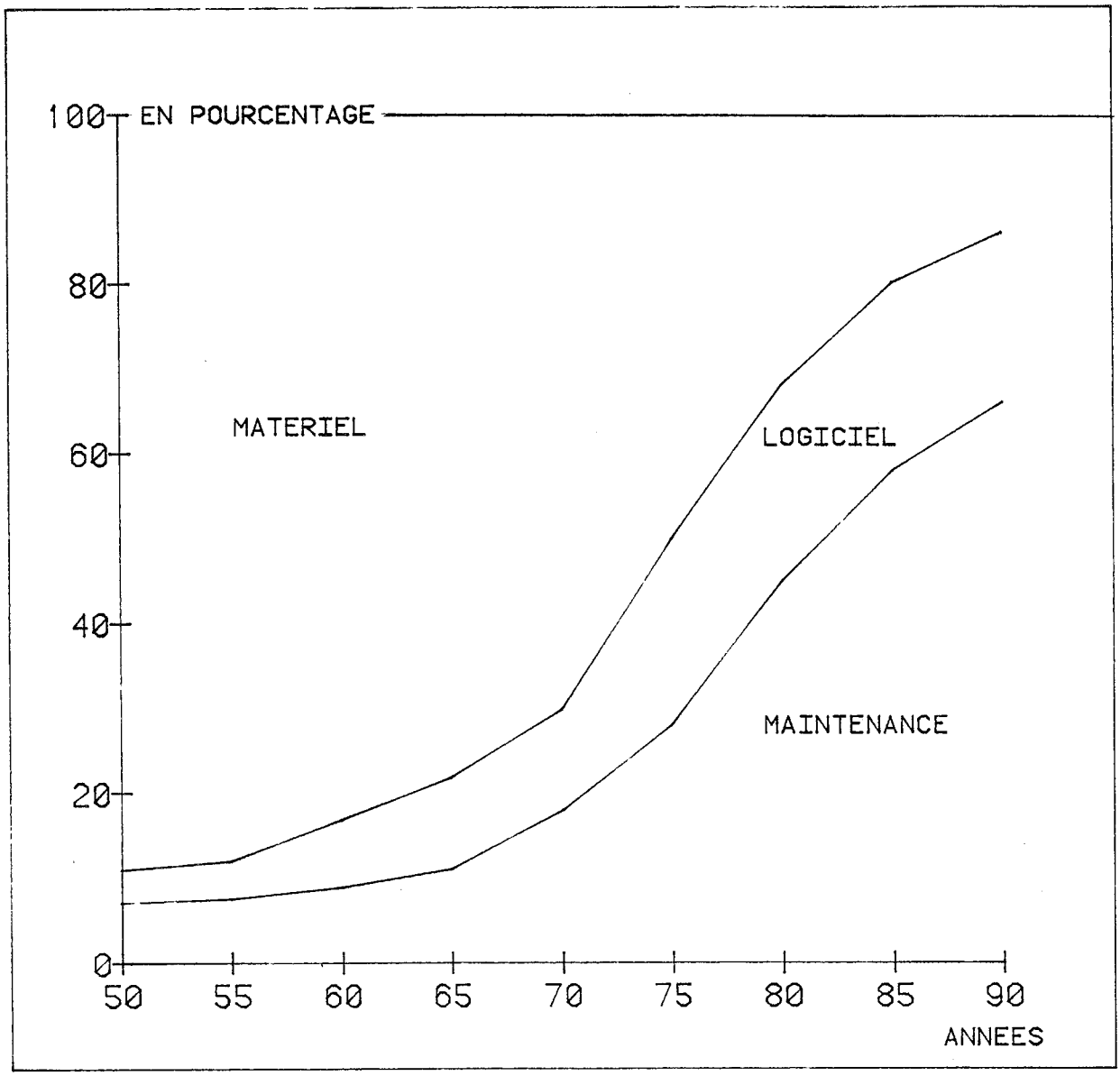
- environnement de programmation pour le langage Modula2
- base de programme ADELE.

9.1 ANNEXE A : COUT DU LOGICIEL

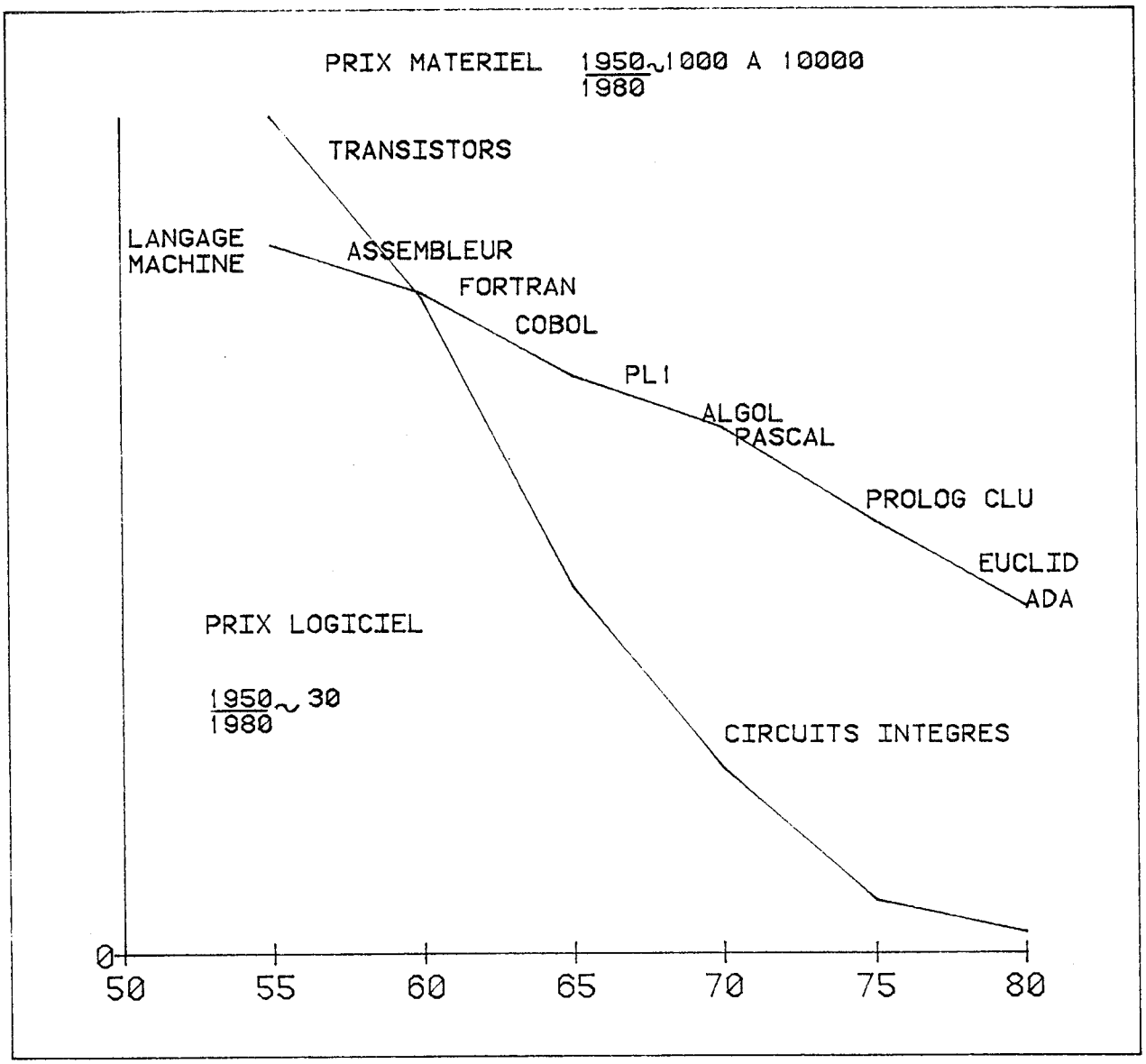
Chacun reconnait que produire du logiciel coûte cher, surtout par rapport au coût du matériel. Les quatre courbes suivantes illustrent bien ces problèmes de coûts. Ces courbes représentent l'évolution des coûts du logiciel de la défense américaine (DOD).



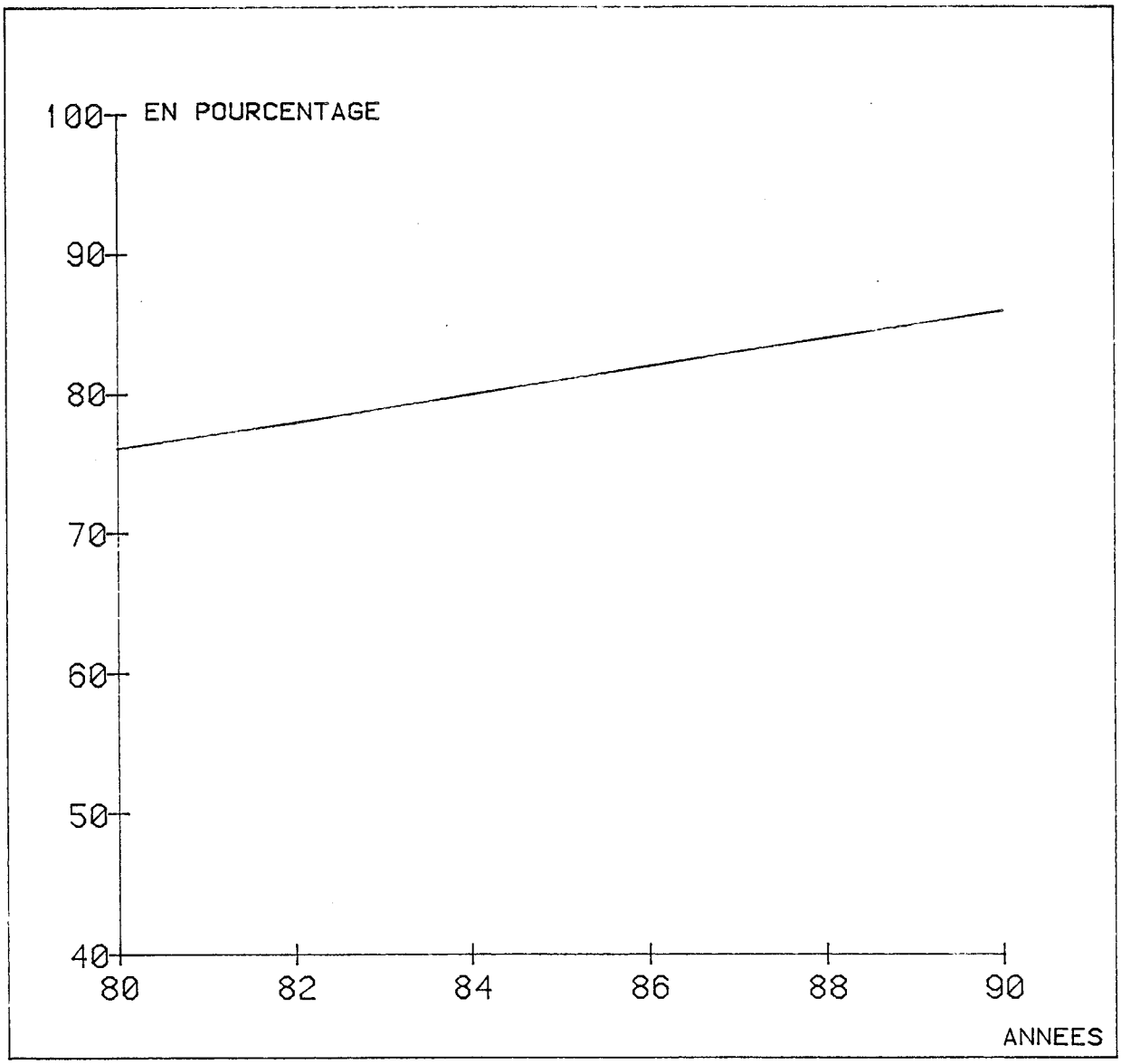
COUT : EVOLUTION DES DEPENSES DU DOD



COUT : EVOLUTION DES POURCENTAGES RESPECTIFS DES COUTS DU MATERIEL ET DU LOGICIEL



COUT : COMPARAISON MATERIEL LOGICIEL



COUT : POURCENTAGE DU COUT DU LOGICIEL
VIS A VIS DU COUT TOTAL
CAS DU D.O.D.

9.2 ANNEXE B - STRUCTURE D'UNE BIBLIOTHEQUE

Les structures des bibliothèques Alis sont normalisées. Une bibliothèque est une collection d'objets de même nature. Elle comprend :

- un ensemble de répertoires. Un répertoire correspond à une version d'un ensemble d'objets. Chaque élément de répertoire contient une identification d'objet et un "pointeur" sur l'objet lui-même.

- les objets eux-mêmes qui sont décrits par une structure de données.

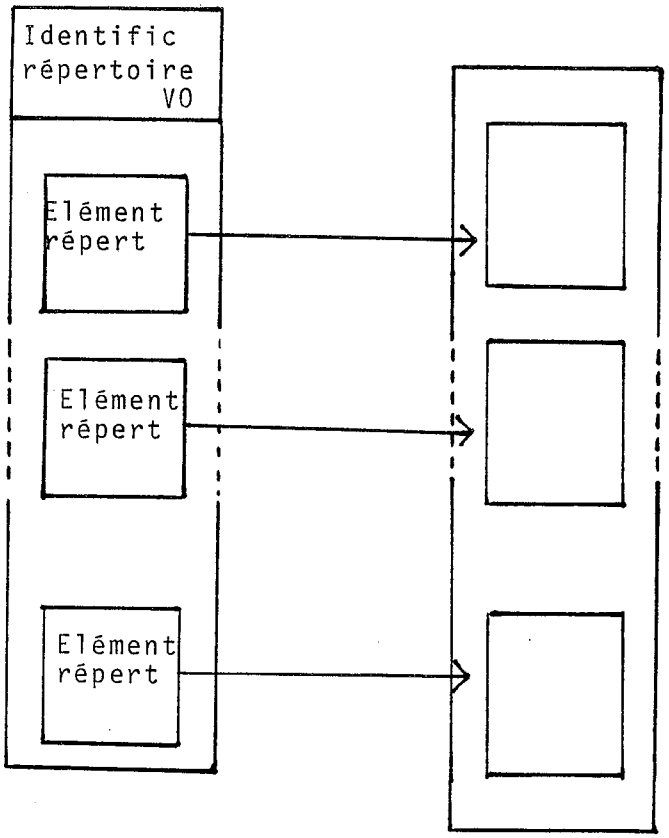
L'accès aux bibliothèques est standardisé et est décrit dans des interfaces.

Le système de répertoire/objet permet de gérer des versions

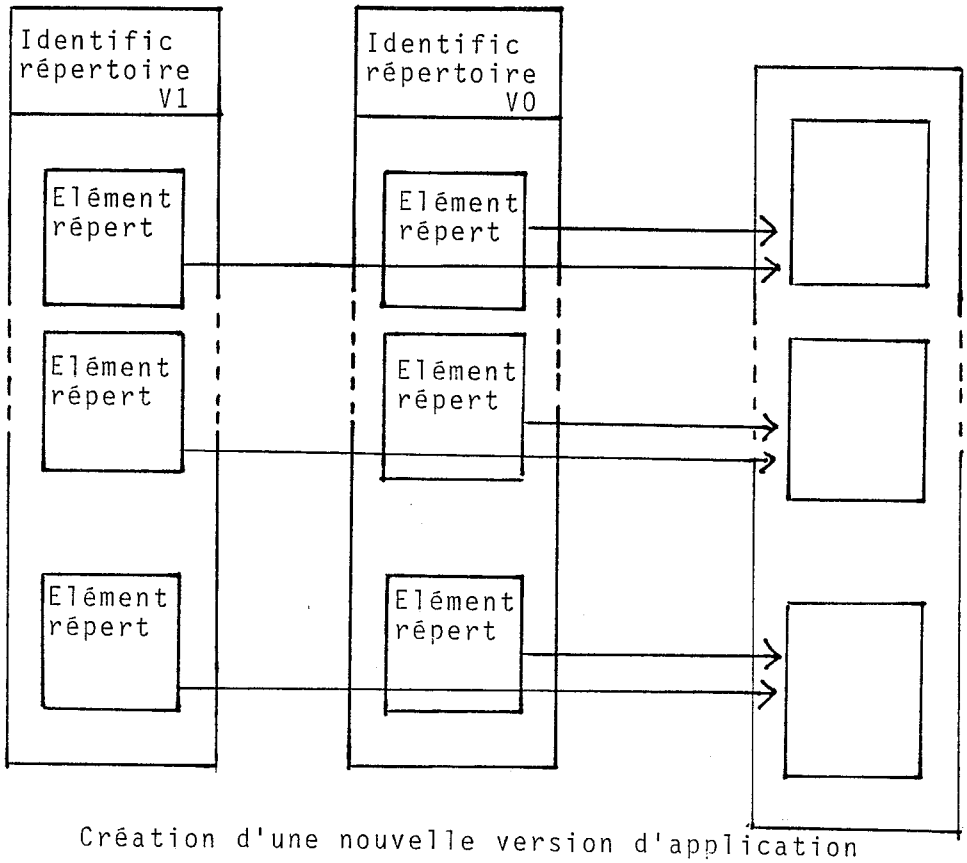
- créer une nouvelle version consiste à dupliquer le répertoire, sans dupliquer les objets (voir schémas pages suivantes).

- L'utilisateur précise la version de répertoire sur laquelle il veut travailler (par défaut la dernière créée).

1

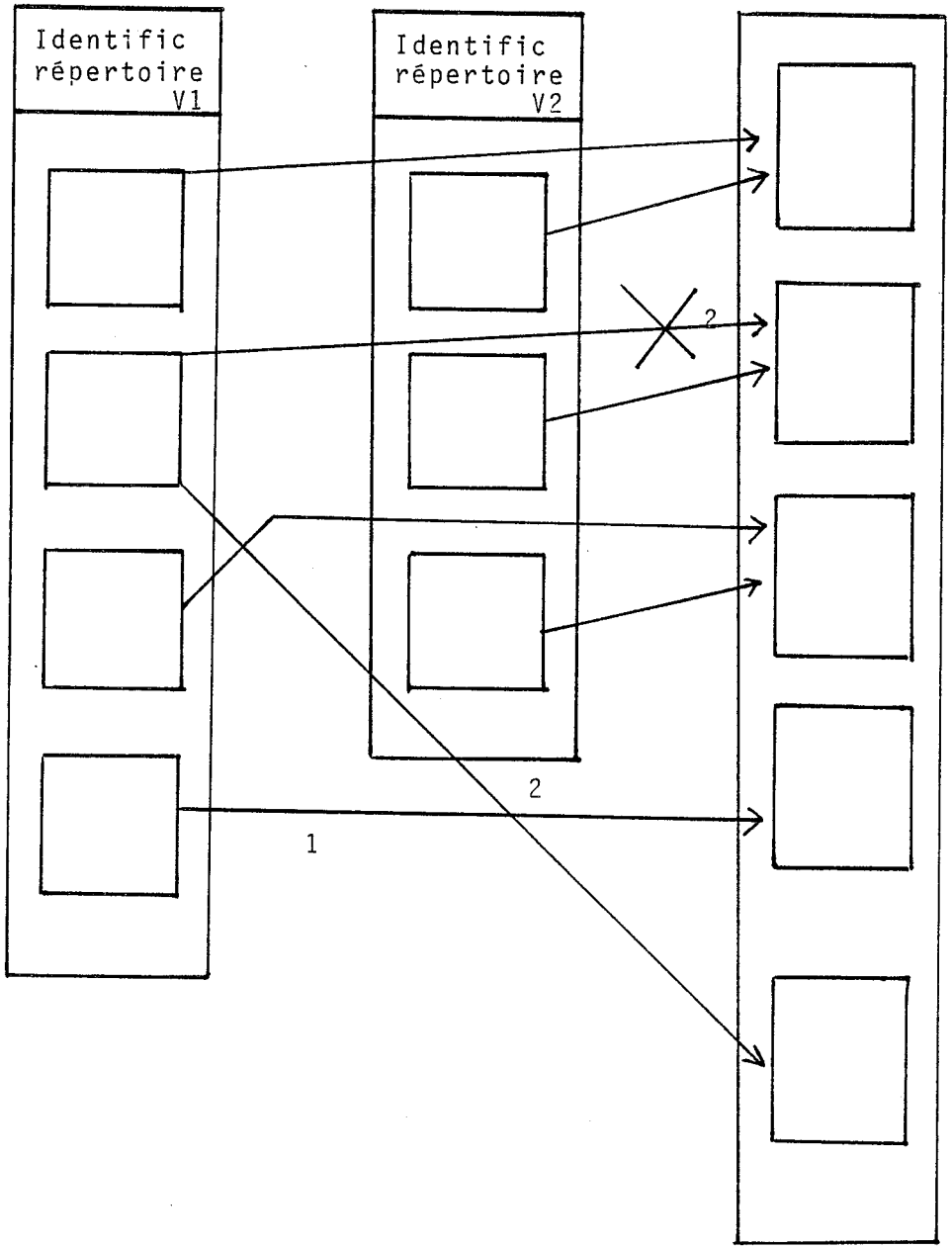


2



Création d'une nouvelle version d'application

3



1 = Ajout d'un nouvel élément dans la version 1
on créé un nouvel élément répertoire et objet

2 = Modification d'un élément dans la version 1
on créé un nouvel élément objet et on modifie
l'élément répertoire

9.3 ANNEXE C : INTERFACE OBJETS LISTE DE COMPOSITION ET OBJETS LIEN

Les objets liste de composition et lien sont décrits par des interfaces permettant de les manipuler. Les interfaces suivants sont ceux permettant de manipuler les objets liste de composition et lien du niveau application,produit,projet.

9.3.1 Interfaces objets liste de composition

- interface objet liste de composition du niveau projet : I:PRO

- interface objet liste de composition du niveau produit : I:PRD

- interface objet liste de composition du niveau application : I:APL

```
PROGRAM INTERFACE I:PRO;
IMPORT PRO;
INTERFACE DATQ,ADQ,IDQ,INTQ,IDFQ;

TYPE
        %*************************************************************************
         *                                                                       *
         * INTERFACE   MODULE PRO : TYPES                                        *
         *                                                                       *
         =*************************************************************************%

        PRO_IDX = 0..32767;
        PRO = RECORD
                LOGIC_NAME : AD$LAB;
                DIRECTORY : AD;
                ID_PROJ : IDFT;
                DIR_PROJ : AD;
                % ARRAY[??] OF ELEMENT_PRO %
                END;
        ELEMENT_PRO = RECORD
                        NOM : T_ID;
                        DATE_DCL : BDATE;
                        %DATE DE DECLARATION%
                        D_APB : APB;
                        %LISTE DES APPLI PUBLIQUES ASSOCIEES%
                        FREE : BOOLEAN;
                        % INDIQUE SI L'ELEMENT EST LIBRE OU NON %
                        END;

PROC
        %*************************************************************************
         *                                                                       *
         * INTERFACE   MODULE PRO : PRIMITIVES EXPORTEES                         *
         *                                                                       *
         =*************************************************************************%

 PROCEDURE PRO$FREE_ELM(VAR ELM_PRO : ELEMENT_PRO;
                        VAR ERREUR : INTEGER;PASCAL;
                        % POSITION D'UN ELEMENT DU BOOLEAN FREE DE ELM_PRO %
 PROCEDURE PRO$STORE(VAR D_PRO:PRO;
                     INDEX:PRO_IDX;
                     VAR ELM_PRO:ELEMENT_PRO;
                     VAR ERREUR : INTEGER);PASCAL;
                     % RANGEMENT INDICE D'UN ELEMENT_PRO %
 PROCEDURE PRO$FETCH(VAR D_PRO:PRO;
                     INDEX:PRO_IDX;
                     VAR ELM_PRO:ELEMENT_PRO;
                     VAR ERREUR : INTEGER);PASCAL;
                     % LECTURE INDICEE D'UN ELEMENT_PRO %
 PROCEDURE PRO$APPEND(VAR D_PRO:PRO;
                      VAR ELM_PRO:ELEMENT_PRO;
```

```
                        VAR ERREUR : INTEGER);PASCAL;
                        %AJOUT  D'UN ELEMENT_PRO %
FUNCTION PRO$EXIST_ELM(VAR D_PRO : PRO;
                         VAR ELM_PRO:ELEMENT_PRO;
                         VAR ERREUR:INTEGER):BOOLEAN;PASCAL;
                         %ACQUISITION SEQUENTIELLE D'UN ELEMENT_PRO%
                         %   FAUX SI PAS D'ELEMENT                  %
PROCEDURE PRO$REWIND(VAR D_PRO:PRO;
                        VAR ERREUR:INTEGER);PASCAL;
                        % REPOSITIONNEMENT DEBUT DE PRO POUR ACCES SEQ %
FUNCTION PRO$SEARCH(VAR D_PRO : PRO;
                      VAR V_ID : T_ID;
                      VAR FIND_ELM : ELEMENT_PRO;
                      VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                      %ACQUISITION DIRECTE D'UN ELEMENT PRO    %
                      %RECHERCHE PAR SON NOM                   %
                      % FAUX SI NON TROUVE                     %
PROCEDURE PRO$UPDATE(VAR D_PRO:PRO;
                       VAR V_ID : T_ID;
                       VAR ELM_PRO:ELEMENT_PRO;
                       VAR ERREUR : INTEGER);PASCAL;
                     % MISE A JOUR D'UN ELEMENT_PRO DANS D_PRO %
                       % SI N'EXISTE PAS ALORS ON AJOUTE A LA FIN  %
                     %       SINON LIBERATION _ REMPLACEMENT %
PROCEDURE PRO$INIT(VAR D PRO:PRO;
                     VAR ERREUR : INTEGER);PASCAL;
                     % CREATION D'UN NOUVEAU DESCRIPTEUR DE  PROJET %
                   %   INITIALISATION DU SUPPORT    %
PROCEDURE PRO$OPEN(VAR D_PRO : PRO;
                     VAR ERREUR : INTEGER);PASCAL;
                     %   OUVERTURE SUPPORT     %
PROCEDURE PRO$CLOSE(VAR D_PRO:PRO;
                      VAR ERREUR : INTEGER);PASCAL;
                      % FERMETURE D'UN DESCRIPTEUR   PROJET %
PROCEDURE PRO$LISTE(VAR D_PRO : PRO;
                      VAR ERREUR : INTEGER);PASCAL;
                      % LISTE D'UN DESCRIPTEUR DE  PROJET %
PROCEDURE PRO$LISTE_ELM(VAR ELM-PRO : ELEMENT-PRO;
                          VAR ERREUR : INTEGER);PASCAL;
                          %LISTE D'UN ELEMENT_PRO %
FUNCTION PRO$EXIST_DEL(VAR D_PRO : PRO;
                         VAR ELM_PRO : ELEMENT_PRO;
                         VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                         % RECHERCHE D'UN ELEMENT PRO DONT LE BOOLEAN FREE = TRU
                         % SI RECHERCHE FRUCTUEUSE ON RENDL'ELEMENT_PRO %
PROCEDURE PRO$LAST_ELM(VAR D_PRO : PRO;
                         VAR INDEX : PRO_IDX;
                         VAR ERREUR : INTEGER);PASCAL;
                         % INDICE DERNIER ELEMENT ACCEDE SEQUENTIELLEMENT  %
PROCEDURE PRO$HIGH_ELM(VAR D_PRO : PRO;
                         VAR INDEX : PRO_IDX;
                         VAR ERREUR : INTEGER);PASCAL;
                         % INDICE DERNIER ELEMENT AJOUTE
PROCEDURE PRO$FREE(VAR D_PRO : PRO;
                     VAR ERREUR : INTEGER);PASCAL;
                     % LIBERATION D'UN DESCRIPTEUR PROJET %
PROCEDURE PRO$CREAT(VAR D_PRO : PRO;
                      VAR ELM_PRO : ELEMENT_PRO;
                      VAR ERREUR : INTEGER);PASCAL;
                      % CREATION D'UN ELEMENT PRO
```

```
PROCEDURE PRO$CLOSE_ELM(VAR ELM_PRO : ELEMENT_PRO;
                        VAR ERREUR : INTEGER);PASCAL;
                        % LIBERATION D'UN ELEMENT PRO %
FUNCTION PRO$INT_PRO(VAR D_PRO : PRO;
                     VAR VINT : T_ID;
                     VAR VPRD,VAPPLI : T_ID;
                     VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                     %VRAI SI INT EST UNE INTERFACE DECLAREE AU NIVEAU PROJET
                     %DANS CE CAS ON REND LE NOM DU PRODUIT %
```

```
PROGRAM INTERFACE I:PRD;
IMPORT PRD
INTERFACE ADQ,IDFQ,INT,ID


TYPE

        %*************************************************************************
        *
        * INTERFACE  MODULE PRD : TYPES
        *
        =*************************************************************************%

       %**ALIS** DECLARATIONS DE TYPES %
       T_TYPAPL = (APUB,APRIV);
       PRD = RECORD
                 LOGIC_NAME : AD$LAB;
                 DIRECTORY : AD;
                 ID PRD : IDFT;
                 DIR_PRD : AD;
                 % ARRAY [??] OF ELEMENT_PRD%
                 END;
       ELEMENT_PRD = RECORD
                        NOM : T_ID;
                        PP : T_TYPAPL;
                        DATE_DCL : BDATE;
                        %DATE DE DECLARATION%
                        D_INT : INT;
                        % INTERFACES/IM ASSOCIEES%
                        FREE : BOOLEAN;
                        % INDIQUE SI L'ELEMENT EST LIBRE OU NON%
                        END;
       PRD_IDX = 0..32767;


PRDC

 %*************************************************************************
  *
  * INTERFACE   MODULE PRD : PRIMITIVES EXPORTEES
  *
  =*************************************************************************%




 PROCEDURE PRD$FREE_ELM(VAR ELM_PRD : ELEMENT_PRD;
                        VAR ERREUR : INTEGER);PASCAL;
                        % POSITIONNEMENT DU BOOLEAN FREE DE ELM_PRD %
 PROCEDURE PRD$STORE(VAR D_PRD:PRD;
                     INDEX:PRD_IDX;
                     VAR ELM PRD:ELEMENT PRD;
                     VAR ERREUR : INTEGER),PASCAL,
                     % RANGEMENT INDICE D'UN ELEMENT-PRD %
```

```
PROCEDURE PRD$FETCH(VAR D_PRD:PRD;
                    INDEX:PRD_IDX;
                    VAR ELM_PRD.ELEMENT_PRD.
                    VAR ERREUR : INTEGER);PASCAL;
                    % LECTURE INDICEE D'UN ELEMENT_PRD %
PROCEDURE PRD$APPEND(VAR D_PRD:PRD;
                     VAR ELM PRD:ELEMENT PRD;
                     VAR ERREUR : INTEGER);PASCAL;
                     %AJOUT D'UN ELEMENT_PRD %
FUNCTION PRD$EXIST_ELM(VAR D_PRD : PRD;
                       VAR ELM_PRD:ELEMENT_PRD;
                       VAR ERREUR:INTEGER):BOOLEAN;PASCAL;
                       %ACQUISITION SEQUENTIELLE D'UN ELEMENT_PRD%
                       %   FAUX SI PAS D'ELEMENT                  %
PROCEDURE PRD$REWIND(VAR D_PRD:PRD;
                     VAR ERREUR:INTEGER);PASCAL;
                     % REPOSITIONNEMENT DEBUT DE PRD POUR ACCES SEQ %
FUNCTION PRD$SEARCH(VAR D_PRD : PRD;
                    VAR V_ID : T_ID;
                    VAR FIND_ELM : ELEMENT_PRD;
                    VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                    %ACQUISITION DIRECTE D'UN ELEMENT PRD    %
                    %RECHERCHE PAR SON NOM                   %
                    % FAUX SI NON TROUVE                     %
PROCEDURE PRD$UPDATE(VAR D_PRD:PRD;
                     VAR V_ID :T_ID;
                     VAR ELM_PRD:ELEMENT_PRD;
                     VAR ERREUR : INTEGER);PASCAL;
                     % MISE A JOUR D'UN ELEMENT_PRD DANS D_PRD
                     %SI N'EXISTE PAS ALORS ON RAJOUTE A LA FIN%
                     %       SINON LIBERATION - REMPLACEMENT%
PROCEDURE PRD$INIT(VAR D_PRD:PRD;
                   VAR ERREUR : INTEGER);PASCAL;
                   % CREATION D'UN NOUVEAU DESCRIPTEUR DE  PROJET %
PROCEDURE PRD$OPEN(VAR D_PRD : PRD;
                   VAR ERREUR : INTEGER);PASCAL;
                   %   OUVERTURE SUPPORT     %
PROCEDURE PRD$CLOSE(VAR D_PRD:PRD;
                    VAR ERREUR : INTEGER);PASCAL;
                    % FERMETURE D'UN DESCRIPTEUR   PROJET %
PROCEDURE PRD$LISTE(VAR D_PRD : PRD;
                    VAR ERREUR : INTEGER);PASCAL;
                    % LISTE D'UN DESCRIPTEUR DE  PROJET %
PROCEDURE PRD$LISTE_ELM(VAR ELM_PRD : ELEMENT_PRD;
                        VAR ERREUR : INTEGER);PASCAL;
                        %LISTE D'UN ELEMENT_PRD %
FUNCTION PRD$EXIST_DEL(VAR D_PRD: PRD;
                       VAR ELM_PRD : ELEMENT_PRD;
                       VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                       % RECHERCHE D'UN ELEMENT PRD DONT LE BOOLEAN FREE = TRU
                       % SI RECHERCHE FRUCTUEUSE ON REND L'ELEMENT_PRD %
PROCEDURE PRD$LAST_ELM(VAR D_PRD : PRD;
                       VAR INDEX : PRD_IDX;
                       VAR ERREUR : INTEGER);PASCAL;
                       % INDICE DERNIER ELEMENT ACCEDE SEQUENTIELLEMENT %
PROCEDURE PRD$HIGH_ELM(VAR D_PRD : PRD;
                       VAR INDEX : PRD-IDX;
                       VAR ERREUR : INTEGER);PASCAL;
                       % INDICE DERNIER ELEMENT AJOUTE%
PROCEDURE PRD$FREE(VAR D_PRD : PRD;
```

```
                        VAR ERREUR : INTEGER);PASCAL;
                        % LIBERATION D'UN DESCRIPTEUR PROJET %
PROCEDURE PRD$CREAT(VAR D_PRD : PRD;
                        VAR ELM_PRD : ELEMENT_PRD;
                        VAR ERREUR : INTEGER);PASCAL;
                        % CREATION D'UN ELEMENT PRD  %
PROCEDURE PRD$CLOSE_ELM(VAR ELM_PRD : ELEMENT_PRD;
                            VAR ERREUR : INTEGER);PASCAL;
                            % LIBERATION D'UN ELEMENT PRD  %
FUNCTION PRD$EXIST_INT(VAR D_PRD : PRD;
                            VAR V_INT : T_ID;
                            VAR V_APPLI : T_ID;
                            VAR V_APPLIO : T_ID;
                            VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                            %VRAI SI VINT EST UNE INTERFACE %
                            %ASSOCIÉE À UNE APPLICATION AUTRE%
                            %QUE L'APPLICATION V_APPLI;     %
                            %SI VRAI ON EN RETOURNE LE NOM
                            %DE L'APPLI DANS V_APPLIO %
FUNCTION PRD$INT_APPLI(VAR D_PRD : PRD;
                            VAR V_INT : T_ID;
                            VAR V_APPLI : T_ID;
                            VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                            %VRAI SI V INT EST UNE INTERFACE DE V_APPLI %
FUNCTION PRD$INT_PRD(VAR D_PRD : PRD;
                        VAR VINT : T_ID;
                        VAR VAPPLI : T_ID;
                        VAR PP : T_TYPINT;
                        VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                        %VRAI SI VINT EST UNE INTERFACE DU PRODUIT PRD%
                        %PP CONTIENT SA NATURE (PRIVEE OU PUBLIQUE) %
                        %ET VAPPLI LE NOM DE L'APPLI ORIGINE%
```

```
PROGRAM INTERFACE I:APL;
IMPORT APL;
INTERFACE ADQ,IDFQ;


TYPE

        %*************************************************************************
         *
         * INTERFACE  MODULE APL : TYPES
         *
         =*************************************************************************%


        APL = RECORD
                  LOGIC_NAME : ADSLAB;
                  DIR_APL : AD;
                  ID_APL : IDFT;
                  D_MDEF : AD;
                  % ARRAY [??] OF ELEMENT_APL%
                  D_MINT : AD;
                  % ARRAY [??] OF ELEMENT_APL%
                  D_MREAL : AD;
                  % ARRAY [??] OF ELEMENT_APL%
                  END;
        APL_IDX = 0..32767;
        ELEMENT_APL = RECORD
                          ID_MOD : IDFT;
                          END;



PROC
          %*************************************************************************
           *
           * INTERFACE   MODULE APL : PRIMITIVES EXPORTEES
           *
           =*************************************************************************


 PROCEDURE APL$FREE_ELM(VAR ELM_APL : ELEMENT_APL;
                        VAR ERREUR : INTEGER);PASCAL;
                        % POSITIONNEMENT DU BOOLEAN FREE DE ELM_APL %
 PROCEDURE APL$STORE(VAR D_APL:APL;
                     INDEX:APL IDX;
                     VAR ELM_APL:ELEMENT_APL;
                     VAR ERREUR : INTEGER);PASCAL;
                     % RANGEMENT INDICE D.UN ELEMENT_APL %
 PROCEDURE APL$FETCH-MDEF(VAR D_APL:APL;
                          INDEX:APL IDX;
                          VAR ELM_APL:ELEMENT_APL;
                          VAR ERREUR : INTEGER);PASCAL;
                          %LECTURE INDICEE D.UN ELEMENT_APL DE TYPE M_DEF%
 PROCEDURE APL$FETCH_MREAL(VAR D_APL:APL;
                           INDEX:APL_IDX;
                           VAR ELM_APL:ELEMENT_APL;
```

```
                                VAR ERREUR : INTEGER);PASCAL;
                                % LECTURE INDICEE D'UN ELEMENT_APL DE TYPE M_REAL%
PROCEDURE APL$FETCH_MINT(VAR D_APL:APL;
                         INDEX:APL_IDX;
                         VAR ELM_APL:ELEMENT_APL;
                         VAR ERREUR : INTEGER);PASCAL;
                         %LECTURE INDICEE D'UN ELEMENT_APL DE TYPE M_INT%
PROCEDURE APL$APPEND(VAR D_APL:APL;
                     VAR ELM APL:ELEMENT APL;
                     VAR ERREUR : INTEGER);PASCAL;
                     %AJOUT  D'UN ELEMENT_APL %
FUNCTION APL$EXIST_ELM_MDEF(VAR D_APL : APL;
                            VAR ELM_APL:ELEMENT_APL;
                            VAR ERREUR:INTEGER):BOOLEAN;PASCAL;
                            %ACQUISITION SEQUENTIELLE D'UN ELEMENT_APL%
                            %DE TYPE M_MDEF,FAUX SI PLUS D'ELEMENT
FUNCTION APL$EXIST_ELM_MINT(VAR D_APL : APL;
                            VAR ELM_APL.ELEMENT_APL.
                            VAR ERREUR:INTEGER):BOOLEAN;PASCAL;
                            %ACQUISITION SEQUENTIELLE D'UN ELEMENT_APL%
                            %DE TYPE M_INT;  FAUX SI PAS ELEMENT %
FUNCTION APL$EXIST_ELM_MREAL(VAR D_APL : APL;
                             VAR ELM_APL:ELEMENT_APL;
                             VAR ERREUR:INTEGER):BOOLEAN;PASCAL;
                             %ACQUISITION SEQUENTIELLE D'UN ELEMENT APL%
                             %DE TYPE M_REAL;  FAUX SI PAS ELEMENT%
                             %   FAUX SI PAS D'ELEMENT %
PROCEDURE APL$REWIND_MDEF (VAR D_APL:APL;
                           VAR ERREUR:INTEGER);PASCAL;
                           % REPOSITIONNEMENT DEBUT DE%
                           %APL M_MDEF POUR ACCES SEQ %
PROCEDURE APL$REWIND_MINT (VAR D_APL:APL;
                           VAR ERREUR:INTEGER);PASCAL;
                           % REPOSITIONNEMENT DEBUT DE%
                           %APL M-MINT POUR ACCES SEQ %
PROCEDURE APL$REWIND_MREAL (VAR D_APL:APL;
                            VAR ERREUR.INTEGER).PASCAL.
                            % REPOSITIONNEMENT DEBUT DE%
                            %APL M_REAL POUR ACCES  SEQ%
FUNCTION APL$SEARCH(VAR D_APL : APL;
                    VAR V_ID : T_ID;
                    VAR V_NAT : T_TYPMOD;
                    VAR FIND_ELM : ELEMENT_APL;
                    VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                    %ACQUISITION DIRECTE D'UN ELEMENT APL     %
                    %RECHERCHE PAR SON NOM ET SA NATURE       %
                    % FAUX SI NON TROUVE                      %
PROCEDURE APL$UPDATE(VAR D_APL:APL;
                     VAR ELM APL:ELEMENT APL;
                     VAR ERREUR : INTEGER);PASCAL;
                     % MISE A JOUR D'UN ELEMENT_APL DANS D_APL %
                     % SI N'EXISTE PAS ALORS ON RAJOUTE A LA FIN%
                     %       SINON LIBERATION - REMPLACEMENT%
PROCEDURE APL$SINIT(VAR D_APL:APL;
                    VAR ERREUR : INTEGER);PASCAL;
                    % CREATION D'UN NOUVEAU DESCRIPTEUR D'APPLICATION %
PROCEDURE APL$OPEN(VAR D_APL : APL;
                   VAR ERREUR : INTEGER);PASCAL;
                   %   OUVERTURE SUPPORT     %
PROCEDURE APL$CLOSE(VAR D_APL:APL;
```

```
                        VAR ERREUR : INTEGER);PASCAL;
                        % FERMETURE D'UN DESCRIPTEUR  D 'APPLICATION%
PROCEDURE APL$LISTE(VAR D_APL : APL;
                        VAR ERREUR : INTEGER);PASCAL;
                        % LISTE D'UN DESCRIPTEUR D'APPLICATION       %
PROCEDURE APL$LIST_ELM(VAR ELM_APL : ELEMENT_APL;
                           VAR ERREUR : INTEGER);PASCAL;
                           %LISTE D'UN ELEMENT_APL %
FUNCTION APL$EXIST_DEL_MDEF (VAR D_APL : APL;
                                VAR ELM_APL : ELEMENT_APL;
                                VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                                % RECHERCHE D'UN ELEMENT APL DE
                                %NATURE M_DEF,DONT LE BOOLEAN FREE EST VRAI%
                                % SI RECHERCHE FRUCTUEUSE ON REND L'ELEMENT_APL%
FUNCTION APL$EXIST_DEL_MINT (VAR D_APL : APL;
                                VAR ELM_APL : ELEMENT_APL;
                                VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                                % RECHERCHE D'UN ELEMENT APL DE %
                                %NATURE M_INT,DONT LE BOOLEAN FREE EST VRAI%
                                % SI RECHERCHE FRUCTUEUSE ON REND L'ELEMENT_APL%
FUNCTION APL$EXIST_DEL_MREAL(VAR D_APL : APL;
                                VAR ELM_APL : ELEMENT_APL;
                                VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                                % RECHERCHE D'UN ELEMENT APL DE   %
                                %NATURE M_REAL,DONT LE BOOLEAN FREE EST VRAI%
                                % SI RECHERCHE FRUCTUEUSE ON REND L'ELEMENT_APL%
PROCEDURE APL$LAST_ELM(VAR D_APL : APL;
                           VAR INDEX_MDEF:APL_IDX;
                           VAR INDEX_MINT:APL_IDX;
                           VAR INDEX_MREAL:APL_IDX;
                           VAR ERREUR : INTEGER);PASCAL;
                           % INDICES DERNIERS ELEMENTS ACCEDES SEQUENTIELLEMENT%
PROCEDURE APL$HIGH_ELM(VAR D_APL : APL;
                           VAR INDEX_MDEF:APL_IDX;
                           VAR INDEX_MINT:APL_IDX;
                           VAR INDEX_MREAL:APL_IDX;
                           VAR ERREUR : INTEGER);PASCAL;
                           % INDICES DERNIERS ELEMENTS AJOUTES      %
PROCEDURE APL$FREE(VAR D_APL : APL;
                      VAR ERREUR : INTEGER);PASCAL;
                      %LIBERATION DES ELEMENTS D'UN DESCRIPTEUR D'APPLI%
```

9.3.2 Interfaces objets lien

- interface objet lien du niveau projet : I:LPO
- interface objet lien du niveau produit : I:LPD
- interface objet lien du niveau application : I:LPA

```
PROGRAM INTERFACE I:LPO;
IMPORT LPO;
INTERFACE ADQ,REM,MAT,IDFQ,ID;


TYPE

        %*************************************************************************
         *                                                                       *
         * INTERFACE  MODULE LPO : TYPES                                         *
         *                                                                       *
         =*************************************************************************%


        LPO=RECORD
                LOGIC_NAME : AD$LAB;
                DIRECTORY : AD;
                ID_PROJ : IDFT;
                %IDENTIFIACTION DU PROJET%
                D_REM_INT : REM;
                %REPERTOIRE DES INTERFACES POUR ACCES A LA MATRICE%
                D_REM_PR : REM;
                %REPERTOIRE DES PRODUITS POUR ACCES A LA MATRICE%
                D_MAT : MAT;
                %MATRICE DES LIENS PRO/INT%
                END;


PROC

        %*************************************************************************
         *
         * INTERFACE   MODULE LPO : PRIMITIVES EXPORTEES
         *
         =*************************************************************************%


 PROCEDURE LPO$CREER(VAR D_LPO : LPO;
                     VAR ERREUR : INTEGER);PASCAL;
                     % INITIALISATION DU SUPPORT%
 PROCEDURE LPO$OPEN(VAR D_LPO : LPO;
                    VAR ERREUR : INTEGER);PASCAL;
                    %OUVERTURE DU SUPPORT%
 PROCEDURE LPO$CLOSE(VAR D_LPO : LPO;
                     VAR ERREUR : INTEGER);PASCAL;
                     %   FERMETURE DU SUPPORT %
 PROCEDURE LPO$MEM(VAR D_LPO : LPO;
                   VAR INT,PROD : T_ID;
                   VAR ERREUR : INTEGER);PASCAL;
                   % MEMORISE LE LIEN ENTRE UN MODULE INTERFACE ET UN PRODUIT%
 PROCEDURE LPO$SUPP(VAR D_LPO : LPO;
                    VAR INT,PROD : T_ID;
                    VAR ERREUR : INTEGER);PASCAL;
                    % SUPPRIME LE LIEN ENTRE UN MODULE INTERFACE ET UN PRODUIT%
 PROCEDURE LPO$INIT_INT_PR(VAR D_LPO : LPO;
                           VAR INT : T_ID;
```

```
                              VAR ERREUR : INTEGER);PASCAL;
                              % INITIALISE LE PARCOURS DES LIENS%
                              %D'UNE INTERFACE AVEC LES PRODUITS %
FUNCTION LPO$INT_PR(VAR D_LPO : LPO;
                    VAR PR : T_ID;
                    VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                    %DELIVRE SEQUENTIELLEMENT TOUS LES PRODUITS UTILISANT %
                    %L'INTERFACE INT DECLAREE DANS LPO$INIT_INT_PR%
PROCEDURE LPO$INIT_PR_INT(VAR D_LPO : LPO;
                              VAR PR : T_ID;
                              VAR ERREUR : INTEGER);PASCAL;
                              % INITIALISE LE PARCOURS DES LIENS%
                              %D'UN PRODUIT AVEC LES INTERFACES %
FUNCTION LPO$PR_INT(VAR D_LPO : LPO;
                    VAR INT : T_ID;
                    VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                    %DELIVRE SEQUENTIELLEMENT TOUTES LES INTERFACES VUES PAR%
                    % LE PRODUIT PR DECLARE DANS LPO$INIT_PR_INT%
PROCEDURE LPO$LISTE(VAR D_LPO : LPO;
                    VAR ERREUR : INTEGER);PASCAL;
                    % LISTE DU SUPPORT   %
PROCEDURE LPO$CREER_INT(VAR D_LPO : LPO;
                        VAR INT : T_ID;
                        VAR ERREUR : INTEGER);PASCAL;
                        %   CREE UN NOUVEL INTERFACE DANS L'OBJET LIEN %
                        %     EN INITIALISANT A NIL TOUS LES LIENS AVEC  %
                        %  LES PRODUITS                                  %
PROCEDURE LPO$CREER_PR(VAR D_LPO : LPO;
                       VAR PR : T_ID;
                       VAR ERREUR : INTEGER);PASCAL;
                       %   CREE UN NOUVEAU PRODUIT DANS L'OBJET LIEN   %
                       %   EN INITIALISANT A NIL TOUS LES LIENS AVEC   %
                       %   LES INTERFACES                              %
PROCEDURE LPO$RAZ_PR(VAR D_LPO : LPO;
                     VAR PR : T_ID;
                     VAR ERREUR : INTEGER);PASCAL;
                     % SUPPRIME TOUS LES LIENS DU PRODUIT PR ,AINSI QUE %
                     % L'ENTREE CORRESPONDANTE %
PROCEDURE LPO$RAZ_INT(VAR D_LPO : LPO;
                      VAR INT : T_ID;
                      VAR ERREUR : INTEGER);PASCAL;
                      %  SUPPRIME TOUS LES LIENS DE L'INTERFACE INT,AINSI QUE%
                      % L'ENTREE CORRESPONDANTE%
```

```
PROGRAM INTERFACE I:LPD;
IMPORT LPD;
INTERFACE ADQ,REM,MAT,ID,IDFQ;


TYPE
        %*************************************************************************
         *
         * INTERFACE  MODULE LPD : TYPES
         *
         =************************************************************************%


        LPD = RECORD
                  LOGIC NAME : AD$LAB;
                  DIRECTORY : AD;
                  ID_PRD : IDFT;
                  %IDENTIFICATION DU PRODUIT%
                  D_REM_INTPB : REM;
                  %REPERTOIRE DES INTERF PUB %
                  D_MAT_PB : MAT;
                  %MATRICE DES LIENS APPLI/INTERF PUB%
                  D_REM_APPLI : REM;
                  %REPERTOIRE DES APPLICATIONS%
                  D_MAT_PV : MAT;
                  %MATRICE DES LIENS APPLI/INTERF PRIVEES%
                  D_REM_INTPV : REM;
                  %REPERTOIRE DES INTERFACES PRIVEES%
                  END;


PROC
        %*************************************************************************
         *                                                                       *
         * INTERFACE   MODULE LPD : PRIMITIVES EXPORTEES                         *
         *                                                                       *
         =************************************************************************%


PROCEDURE LPDSCREER(VAR D_LPD : LPD;
                    VAR ERREUR : INTEGER);PASCAL;
                    % INITIALISATION DU SUPPORT%
PROCEDURE LPDSOPEN(VAR D_LPD : LPD;
                   VAR ERREUR : INTEGER);PASCAL;
                   %OUVERTURE DU SUPPORT%
PROCEDURE LPDSCLOSE(VAR D_LPD : LPD;
                    VAR ERREUR : INTEGER);PASCAL;
                    %   FERMETURE DU SUPPORT%
PROCEDURE LPDSPEN(VAR D_LPD : LPD;
                  VAR INT : T_ID;
```

```
                    VAR TINT : T_TYPINT;
                    VAR APPLI : T_ID;
                    VAR ERREUR : INTEGER);PASCAL;
                    % MEMORISE LE LIEN ENTRE UN MODULE %
                    %INTERFACE ET UNE APPLICATION      %
PROCEDURE LPD$SUPP(VAR D_LPD : LPD;
                    VAR INT : T_ID;
                    VAR TINT : T_TYPINT;
                    VAR APPLI : T_ID;
                    VAR VIDE : BOOLEAN;
                    VAR ERREUR : INTEGER);PASCAL;
                    %SUPPRIME LE LIEN ENTRE UN MODULE%
                    %INTERFACE ET UNE APPLICATION
                    %VIDE = VRAI S'IL N'Y A PLUS DE LIENS%
                    % AVEC L'INTERFACE
PROCEDURE LPD$INIT_INT_APPLI(VAR D_LPD : LPD;
                             VAR INT : T_ID;
                             VAR TINT : T_TYPINT;
                             VAR ERREUR : INTEGER);PASCAL;
                             % INITIALISE LE PARCOURS DES LIENS%
                             %D'UNE INTERFACE / APPLICATION    %
FUNCTION LPD$INT_APPLI_PB(VAR D_LPD : LPD;
                          VAR APPLI : T_ID;
                          VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                          %DELIVRE SEQUENTIELLEMENT TOUTES LES
                          %APPLICATIONS UTILISANT L'INTERFACE%
                          %PUBLIQUE INT DECLAREE DANS LPD$INIT_INT_APPLI%
FUNCTION LPD$INT_APPLI_PV(VAR D_LPD : LPD;
                          VAR APPLI : T_ID;
                          VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                          %DELIVRE SEQUENTIELLEMENT TOUTES LES %
                          %APPLICATIONS UTILISANT L'INTERFACE  %
                          %PRIVEE INT DECLAREE DANS LPD$INIT_INT_APPLI%
PROCEDURE LPD$INIT_APPLI_INT(VAR D_LPD : LPD;
                             VAR APPLI : T_ID;
                             VAR ERREUR : INTEGER);PASCAL;
                             % INITIALISE LE PARCOURS DES LIENS%
                             % D'UNE APPLICATION / INTERFACES  %
FUNCTION LPD$APPLI_INT_PB(VAR D_LPD : LPD;
                          VAR INT : T_ID;
                          VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                          %DELIVRE SEQUENTIELLEMENT TOUTES%
                          %LES INTERFACES PUBLIQUES VUES PAR %
                          % L' APPLICATION APPLI DECLAREE %
                          % DANS LPD$INIT_APPLI_INT
FUNCTION LPD$APPLI_INT_PV(VAR D_LPD : LPD;
                          VAR INT : T_ID;
                          VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                          %DELIVRE SEQUENTIELLEMENT TOUTES%
                          %LES INTERFACES PRIVEES VUES PAR%
                          % L' APPLICATION APPLI DECLAREE%
                          %DANS LPD$INIT_APPLI_INT       %
PROCEDURE LPD$LISTE(VAR D_LPD : LPD;
                    VAR ERREUR : INTEGER);PASCAL;
                    % LISTE DU SUPPORT   %
PROCEDURE LPD$CREER_INT(VAR D_LPD : LPD;
                        VAR INT : T_ID;
                        VAR TINT : T_TYPINT;
                        VAR ERREUR : INTEGER);PASCAL;
                        %   CREE UN NOUVEL INTERFACE DANS L'OBJET LIEN %
```

```
                          %    EN INTIALISANT A NIL TOUS LES LIENS AVEC  %
                          %  LES APPLICATIONS
PROCEDURE LPD$CREER_APPLI(VAR D_LPD : LPD;
                            VAR APPLI : T_ID;
                            VAR ERREUR : INTEGER);PASCAL;
                            %   CREE UN NOUVEAU APPLICATION DANS L'OBJET LIEN
                            %   EN INITIALISANT A NIL TOUS LES LIENS AVEC
                            %   LES INTERFACES
PROCEDURE LPD$RAZ_APPLI(VAR D_LPD : LPD;
                          VAR APPLI : T_ID;
                          VAR ERREUR : INTEGER);PASCAL;
                          %SUPPRIME TOUS LES LIENS D'UNE %
                          %APPLICATION APPLI,AINSI QUE   %
                          %   L'ENTREE CORRESPONDANTE %
PROCEDURE LPD$RAZ_INT(VAR D_LPD : LPD;
                        VAR INT : T_ID;
                        VAR TINT : T_TYPINT;
                        VAR ERREUR : INTEGER);PASCAL;
                        %  SUPPRIME TOUS LES LIENS DE%
                        %L'INTERFACE INT,ANSI QUE    %
                        % L'ENTREE CORRESPONDANTE
```

```
PROGRAM INTERFACE I:LPA;
IMPORT LPA;
INTERFACE REM,MAT,IDFT,ADQ,ID;


TYPE
        %*************************************************************************
         *
         * INTERFACE  MODULE LPA : TYPES
         *
         =*************************************************************************%

        LPA = RECORD
                  LOGIC_NAME : AD$LAB;
                  ID_APPLI : IDFT;
                  %IDENTIFIACTION DE L'APPLI%
                  DIR_LPA : AD;
                  D REM APL : REM;
                  %REPERTOIRE DES MODULES DEF + REAL +INT DE L'APPLI%
                  MAT_IMP : MAT;
                  %MATRICE DES IMPORT %
                  D_REM_INT : REM;
                  %REPERTOIRE DES MODULES INTERFACES EXTERNES%
                  MAT_INT : MAT;
                  %MATRICE DES INTERFACES%
                  END;


PROC
       %*************************************************************************
         *
         * INTERFACE   MODULE LPA : PRIMITIVES EXPORTEES                        *
         *                                                                      *
         =*************************************************************************%


                          %**ALIS** ENTETES DES PRIMITIVES EXPORTEES %
PROCEDURE LPA$CREER(VAR D_LPA : LPA;
                    VAR ERREUR : INTEGER);PASCAL;
                    % INITIALISATION DU SUPPORT%
PROCEDURE LPA$OPEN(VAR D_LPA : LPA;
                   VAR ERREUR : INTEGER);PASCAL;
                   %OUVERTURE DU SUPPORT%
PROCEDURE LPA$CLOSE(VAR D_LPA : LPA;
                    VAR ERREUR : INTEGER);PASCAL;
                    %   FERMETURE DU SUPPORT%
PROCEDURE LPA$MEM_IMP(VAR D_LPA : LPA;
                      VAR MODL : T_ID;
                      VAR VMAT : T_TYPMOD;
                      VAR MDEF : T_ID;
                      VAR ERREUR : INTEGER);PASCAL;
                      % MEMORISE LE LIEN IMPORT ENTRE %
                      %UN MODULE ET UN MODULE DEFINITION%
PROCEDURE LPA$MEM_INT(VAR D_LPA : LPA;
                      VAR MODL : T_ID;
```

```
                        VAR VNAT : T_TYPMOD;
                        VAR MINT : T_ID;
                        VAR ERREUR : INTEGER);PASCAL;
                        % MEMORISE LE LIEN INTERFACE ENTRE %
                        % UN MODULE ET UN MODULE INTERFACE %
PROCEDURE LPA$SUPP_IMP(VAR D_LPA : LPA;
                        VAR MODL : T_ID;
                        VAR VNAT : T_TYPMOD;
                        VAR MDEF : T_ID;
                        VAR ERREUR : INTEGER);PASCAL;
                        % SUPPRIME LE LIEN IMPORT ENTRE %
                        %UN MODULE ET UN MODULE DEFINITION%
PROCEDURE LPA$SUPP_INT(VAR D_LPA : LPA;
                        VAR MODL : T_ID;
                        VAR VNAT : T_TYPMOD;
                        VAR MINT : T_ID;
                        VAR N_LIEN : BOOLEAN;
                        VAR ERREUR : INTEGER);PASCAL;
                        % SUPPRIME LE LIEN INTERFACE ENTRE %
                        %UN MODULE ET UN MODULE INTERFACE
                        %S'IL N'Y A PLUS DE LIENS AVEC LE %
                        %MODULE INTERF ALORS N_LIEN EST VRAI%
PROCEDURE LPA$INIT_MDEF_MODL(VAR D_LPA : LPA;
                              VAR MDEF : T_ID;
                              VAR ERREUR : INTEGER);PASCAL;
                              % INITIALISE LE PARCOURS DES LIENS %
                              % ENTRE UN MODULE DEFINITION ET LES %
                              % MODULES QUI L'IMPORTENT          %
                              %  LE MODULE DEF MDEF%
FUNCTION LPA$MDEF_MODL(VAR D_LPA : LPA;
                        VAR MODL : IDFT;
                        VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                        % DELIVRE SEQUENTIELLEMENT TOUS LES MODULES IMPORTANT %
                        % LE MODULE MDEF DECLARE DANS LPA$INIT_MDEF_MODL %
PROCEDURE LPA$INIT_MODL_MDEF(VAR D_LPA : LPA;
                              VAR MODL : T_ID;
                              VAR VNAT : T_TYPMOD;
                              VAR ERREUR : INTEGER);PASCAL;
                              % INITIALISE LE PARCOURS DES
                              %LIENS IMPORT D'UN MODULE     %
FUNCTION LPA$MODL_MDEF(VAR D_LPA : LPA;
                        VAR MDEF : IDFT;
                        VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                        % DELIVRE SEQUENTIELLEMENT TOUS LES MODULES%
                        %DEFINITION IMPORTES PAR LE
                        % MODULE DECLARE DANS LPA$INIT_MODL_MDEF %
PROCEDURE LPA$INIT_MINT_MODL(VAR D_LPA : LPA;
                              VAR MINT : T_ID;
                              VAR ERREUR : INTEGER);PASCAL;
                              % INITIALISE LE PARCOURS DES LIENS%
                              % ENTRE   LE MODULE INTERFACE MINT%
                              %ET LES MODULES QUI L'INTERFACENT%
FUNCTION LPA$MINT_MODL(VAR D_LPA : LPA;
                        VAR MODL : IDFT;
                        VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                        % DELIVRE SEQUENTIELLEMENT TOUS LES MODULES INTERFACANT
                        % LE MODULE MINT DECLARE DANS LPA$INIT_MINT_MODL %
PROCEDURE LPA$INIT_MODL_MINT(VAR D_LPA : LPA;
                              VAR MODL : T_ID;
                              VAR VNAT : T_TYPMOD;
```

```
                                VAR ERREUR : INTEGER);PASCAL;
                                % INITIALISE LE PARCOURS DES LIENS %
                                %INTERFACE DU MODULE MODL
FUNCTION LPA$MODL_MINT(VAR D_LPA : LPA;
                       VAR MINT : IDFT;
                       VAR ERREUR : INTEGER):BOOLEAN;PASCAL;
                       %  DELIVRE SEQUENTIELLEMENT TOUS %
                       %LES MODULES INTERFACES IMPORTES PAR%
                       %LE MODULE DECLARE DANS LPA$INIT_MODL_MINT%
PROCEDURE LPA$LISTE(VAR D_LPA : LPA;
                    VAR ERREUR : INTEGER);PASCAL;
                    % LISTE DU SUPPORT   %
PROCEDURE LPA$CREER_MODL(VAR D_LPA : LPA;
                         VAR MODL : IDFT;
                         VAR ERREUR : INTEGER);PASCAL;
                         %   CREE UN NOUVEAU MODULE DANS L'OBJET LIEN %
                         %  EN INTIALISANT A NIL TOUS SES LIENS%
PROCEDURE LPA$CREER_INTEX(VAR D_LPA : LPA;
                          VAR INTEX : IDFT;
                          VAR ERREUR : INTEGER);PASCAL;
                          %CREE UN NOUVEAU INTERFACE EXTERNE%
                          %DANS L'OBJET LIEN
                          %EN INITIALISANT A NIL TOUS SES LIENS %
PROCEDURE LPA$RAZ_MODL(VAR D_LPA : LPA;
                       VAR MODL : T_ID;
                       VAR VNAT : T_TYPMOD;
                       VAR ERREUR : INTEGER);PASCAL;
                       % SUPPRIME TOUS LES LIENS D'UN MODULE ,AINSI QUE %
                       % L'ENTREE CORRESPONDANTE%
PROCEDURE LPA$RAZ_INT(VAR D LPA : LPA;
                      VAR INT : T_ID.
                      VAR ERREUR : INTEGER);PASCAL;
                      %SUPPRIME TOUS LES LIENS DE%
                      %L'INTERFACE EXTERNE INT,AINSI QUE%
                      % L'ENTREE CORRESPONDANTE %
```

9.4 ANNEXE D : PRESENTATION DE QUELQUES ATELIERS

9.5 EM2 : UN ENVIRONNEMENT POUR MODULA2

Modula2 est un langage issu du Pascal supportant la compilation séparée des modules. A ce titre il est très proche du Pascal/S développé dans le cadre d'Alis. Ce langage est très largement utilisé à l'Université de Genève qui a développé un environnement de programmation autour de Modula2.

D'autre part l'environnement EM2 est très proche de l'environnement Pascal/S pour ce qui est de la gestion des liens,avec une particularité intéressante : l'utilisation d'une base de données relationnelle.

Les objectifs visés sont les suivants :

- petits projets
- gestion de la complexité des interactions entre modules
- possibilité de récupérer les outils existant pour les intégrer à l'environnement
- ouverture vers de futurs développements d'outils ou de langages
- portabilité de l'environnement

9.5.0.1 Présentation de Modula2

Le langage Modula2 [WIR 82] Est un langage issu du Pascal auquel il emprunte la plupart de ses structures de données et de contrôle.Il offre par ailleurs de nombreuses améliorations par rapport à Pascal : accès à des adresses absolues en mémoire, type mot mémoire, manipulation de bit,etc... Ces extensions permettent l'écriture de systèmes d'exploitation et de noyaux systèmes.

La notion de module introduite dans Modula2 est un concept de structuration de programmes. Ces modules permettent de regrouper des objets (types, variables, constantes, procédures) de manière cohérente. Ils permettent ainsi, comme Pascal/S par son mécanisme d'interface, de définir des types abstraits. Chaque module est composé d'un module de définition où sont décrits les objets exportés par le module et d'un module d'implantation décrivant les algorithmes de traitement. Un module peut importer des objets définis et exportés par un autre module, dans ce cas le premier module devient dépendant du second. L'ensemble des dépendances entre modules constitue le graphe des dépendances du programme. Le graphe ne doit pas comporter de cycle. Tout programme Modula2 doit posséder un module dont aucun autre ne dépend (qui n'exporte rien), ce module est appelé racine du programme.

Modula2 supporte la compilation séparée des modules. Ce mécanisme permet donc de modifier un programme sans avoir à recompiler tous les modules, sans toutefois perdre la notion de vérification des types entre modules. Le maintien de la cohérence d'un programme exige que tout module soit compilé après chaque module définition dont il dépend. Comme pour Pascal/S ce maintien de la cohérence pose des problèmes lorsque le nombre de modules devient important et leurs interdépendances complexes.

9.5.0.2 Environnement EM2

EM2 offre une méthode de développement de programmes en Modula2 et des outils. La méthode utilise simultanément les approches "Bottom-up" et "Top-down".

L'approche "Bottom-up" consiste à définir les objets de base utilisés dans le système et à construire par regroupements et généralisations successifs des modules d'un niveau d'abstraction toujours plus élevé (types abstraits). De cette manière on obtient un ensemble de modules caractérisés par leur cohérence de communications (modules gérant chacun un type de donnée particulier).

L'approche "Top-down" est basée sur deux types de décomposition du système : décompposition fonctionnelle et décomposition par flots de données. La décomposition fonctionnelle amène à définir des modules réalisant certaines fonctions du système, ils rendent un résultat à partir d'une donnée fournie. La décomposition par flots de données amène à définir des modules interconnectés qui reçoivent leurs données d'un ou plusieurs modules prédécesseurs et transmettent les résultats de leurs traitements à un ou plusieurs modules successeurs.

L'ensemble des spécifications des modules forme le squelette de l'application, et à partir de ces spécifications on peut écrire les modules de définition qui servent à définir les objets exportés visibles de chaque module (types, procédures, constantes, variables).

La décomposition ainsi obtenue n'est pas nécessairement définitive. Elle est amenée à évoluer dans le temps. C'est pourquoi EM2 autorise des modifications dans le découpage en modules.

EM2 fournit un ensemble intégré d'outils possédant un certain nombre de caractéristiques communes.

- interaction standard avec l'usager

- vision identique des objets supportés par l'environnement (de type base de données relationnelle).

EM2 fournit classiquement un compilateur, un éditeur de liens, un moniteur d'exécution, mais aussi :

- un éditeur syntaxique

- un analyseur de structure : cet outil est le gestionnaire des liens entre modules, concernant l'ordre des compilations, la recompilation et l'édition de liens.

- gestionnaire de version.

Ces outils appelés outils de base de l'environnement EM2 sont les seuls à mettre à jour la base de données support de l'environnement.

Les outils complémentaires consultent simplement la base de données. Il a été notamment développé un outil de visualisation de la structure Modulaire d'un programme.

9.5.0.3 La base de donnés support de l'environnement

EM2 utilise une base de donnée relationnelle comme support de l'environnement, ce qui permet entre autre de répondre à l'objectif de vision identique des objets de l'environnement pour les outils.

EM2 propose le modèle relationnel suivant :

- les informations sont : projet, version, module (définition, implantation,racine), objet de Modula2, Importation, bibliothèque de modules.

- le schéma de base de données relationnel est :

Projet (nom-projet, Date-début, nom-fichier-documentation, catégorie)

Module (nom-module, nom-projet, type-module, nom fichier-source, Date-édition des liens, Date-analyse-structure, Date-compilation)

Objet (nom-objet, nom-module, nom-projet, classe-objet, première-ligne-objet, dernière-ligne-objet)

Importe (nom-module, nom-projet, type-module, nom-module-importé, nom-projet-importé)

Version (numéro-version, nom-projet, nom-module)

Librairie (nom-projet, nom-projet-librairie, ordre-de-recherche)

Edition-de-lien (nom-module, nom-projet, Date-édition-de-lien, Fichier-édition-de-lien, fichier-exécutable)

Où

Catégorie = catégorie d'un projet (privé, bibliothèque)

Type-module = définition, implantation, racine

Classe-objet = classe d'un objet (type, constante, variable, procédure)

Première-ligne-objet et dernière-ligne-objet ligne du texte source où se trouve la définition d'un objet.

Ordre-de-recherche numéro d'ordre d'une bibliothèque dans la recherche des objets importés par les modules.

La base de données ne contient donc pas physiquement les programmes des projets mais plutôt une description de leur contenu sémantiquement utile à l'environnement. Elle constitue le niveau logique par rapport au niveau physique constitué par le système de fichiers. EM2 assure la correspondance directe entre l'état des fichiers et l'état de la base de données. Ainsi la base de données doit être mise à jour chaque fois qu'un événement modifie un des fichiers (édition, compilation). Les modifications les plus profondes peuvent survenir après la modification d'un fichier source par l'éditeur de texte : modification de la structure des dépendances entre modules. EM2 fournit donc un outil d'analyse de structure capable d'analyser les modules d'un programme afin de mettre à jour les relations objet, module, et importe de la base de donnée. Les autres outils de l'environnement modifiant les fichiers (compilation, éditions des liens) n'ont pas d'influence sur la structure logique des programmes : la mise à jour de la base de données est simple (par exemple modifications des dates dans les relations module et édition des biens).

9.6 LA BASE DE PROGRAMME ADELE

ADELE est un projet développé à l'Imag Grenoble. Il s'agit d'un atelier logiciel dans lequel les programmes sont construits selon un processus de décomposition modulaire. Adèle offre un intéressant système de gestion de versions. Dans ADELE les entités et les relations qui existent entre elles sont prédéfinies.

ENTITES : ce sont les suivantes :

- Interface , environ : une interface décrit les ressources qu'une réalisation offre à ses utilisateurs. Une interface contient des déclarations de constantes, types, variables, procédures et fonctions. Un environ ne contient que des déclarations de types et de constantes.

- famille : une famille définit une ensemble d'interfaces "voisines" et logiquement reliées entre elles.

- réalisation : une réalisation est un (ou plusieurs) programme qui réalise les ressources fournies par l'interface associée. Une version de réalisation définit une réalisation particulière. Chaque version peut subir une série de révisions désignées par des numéros entiers successifs.

Pour chacune de ces entités on a les fichiers suivants :

- Texte : il s'agit du texte source du programme pour une réalisation des déclarations pour une interface.

- Documentation : Document décrivant l'entité associée.

- Manuel : information interprétable par le système, fixant les contraintes d'utilisation de l'objet associé. Ces contraintes portent essentiellement sur la visibilité entre familles et sur la construction de réalisations composées de plusieurs autres réalisations. Il sert à gérer des versions.

Relation

Les relations entre entités sont celles définies par la décomposition Modulaire, et d'autres sont propres aux règles de développement utilisees dans ADELE. Les relations dues à la décomposition Modulaire sont les suivantes :

- Relation de dépendance entre une réalisation et l'interface qu'elle utilise.

- Relation implémente : entre une interface et une de ses réalisations associées

- Relation composition : entre une réalisation et l'ensemble des réalisations qui la composent.

Etapes de la construction du logiciel dans Adele

La construction d'un logiciel passe par plusieurs étapes : son découpage en composants, la réalisation de chaque composant, et la constitution du logiciel par assemblage des composants.

Décomposition d'un logiciel

Les entités définies dans Adèle influent sur la façon dont le compteur va décomposer un logiciel :

- Préciser l'interface de ce logiciel

- Préciser la réalisation associée :

  - cette réalisation est simple (réalisable par un seul programme)

  - cette réalisation est complexe : elle est divisée en plusieurs logiciels. Le même algorithme doit être répété pour chacun d'eux.

On obtient ainsi une arborescence d'interfaces, à laquelle correspond une arborescence de famille (une famille pour chaque interface); chaque famille possède 0 ou n familles filles et une seule famille mère.

Utilisation des familles

Une réalisation ou une interface peut utiliser les ressources d'une interface. Dans Adele deux mécanismes le permettent :

- Importation : une réalisation peut importer des interfaces. Les ressources de l'interface importée sont partagées entre toutes les réalisations qui utilisent la même famille.

- Inclusion : une interface ou environ peut inclure des environs (au sens lexicographique). Un environ peut en inclure d'autres et récursivement

Visibilité entre familles

Les réalisations et interface d'une famille peuvent utiliser les interfaces et environ :

- des familles filles

- d'une famille nom fille en créant un lien vers celle-ci, dite externe. La famille externe ainsi liée est alors utilisée comme une famille fille.

Cependant, par sécurité, une famille doit autoriser son utilisation par des familles autre que son père. Cette autorisation peut être collective (toute une branche de l'arborescence) ou nominative (une famille en particulier). L'autorisation porte sur toutes les interfaces de la famille (par défaut) ou nominativement sur certaines interfaces seulement. L'arborescence des familles devient donc un grape orienté, sans cycle.

Composition du logiciel

Adèle fournit un intéressant mécanisme permettant de reconstituer un logiciel à partir de ses composants (famille, interface) et d'assurer la cohérence du développement des logiciels. Pour cela a été introduit dans Adele la notion de Manuel associée à une réalisation. Un manuel R comprend les rubriques suivantes :

- Rubrique attributs ("attr"). Les attributs définissent les propriétés qui caractérisent la réalisation.

- Rubrique sélection ("sel") : qui permet la désignation des réalisations qui devront faire partie de la liste de composition de toute réalisation composée incluant R. Les désignations possibles sont les suivantes : désignation nominative c'est à dire que le nom des réalisations à choisir est indiquée, désignation par attribut c'est à dire que les réalisations choisies sont celles dont le ou les attributs correspondent à ceux indiquées :

- Rubrique défaut ("def") : désignation par défaut qui est prise en compte si aucune sélection ("sel") ne porte sur la famille désignée.

Exemple

```
Manuel >ADL>base-i2-v1

   /*manuel associé à la réalisation V1 de l'interface i2 de la famille
           >ADL>base*/

Attr /*attributs de cette réalisation*/

          Auteur = pierre

          Type = debug

          Version = demo3

Def /* pas de désignation par défaut */

Sel /* Sélection */

Acces-i1-v2 /* Cette réalisation est imposée %

End
```

Le manuel d'une réalisation composée est l'expression des contraintes que doivent vérifier les réalisations de sa liste de composition. Ces contraintes sont les rubriques "sel" et "def" des manuels

Exemple :

```
Manuel Realcomp > ADL-i3-vdemo3

          /* Manuel de la réalisation composée vdemo3 */

Def

          > * (Trace = nom) /* pour toutes les familles filles choisie de
          préférence pas de trace */

Sel

          > ADL-i3-Startdemo3 /* réalisation startdemo imposée */

          > * (Type = demo3) /* pour toutes les familles prendre une
          réalisation ayant l'attribut type = demo3 */

End
```

Evolution des logiciels dans Adèle

La forte interdépendance des entités entre elles due aux notions de relations, composition entraine qu'une modification d'un composant entraine des répercussions parfois complexes.

Modification d'une réalisation

Dans la mesure où la nouvelle réalisation respecte les définitions de l'interface à laquelle elle est associée, la modification est logiquement transparente aux utilisateurs, si ce n'est que les codes objets incluant cette réalisation sont périmés.

Modification d'une interface

Si une interface est modifiée, les réalisations associées et celles qui en dépendent directement deviennent incohérentes.

Modification d'un manuel

La modification des rubriques "Sel" et "def" d'un manuel peut induire des modifications de la liste de composition de certaines réalisations composées. Celles-ci sont donc à reconstruire, et ceci de façon récursive.

10 BIBLIOGRAPHIE

AL 83 Alis : méthodologie Alis
Note interne Sems . Doussy j.

CAD 81 Cades - sofware engineering in practice 1979 IEE
Proc. 4th International Conf. On Soft. Eng. , Munich , sept 79

CAS 81 Cashin p.m, Joliat m.l,Kamel r.f, Lasher d.m.
Experience with a Modular typed langage : PROTEL
Proc. 5th international conf. On soft. Eng. San Diego March 1981

CHV 82 Cheval j.l.,Estublier j.,Ghould s.,Krakowiak s.
Modularité et composition de programmes dans l'atelier logiciel Adele
Colloque génie logiciel ,Paris 1982

FEL 79 Feldman s.i.
Make : a program for maintaining software
Software practice and experience ,Vol. 9,pp 255-265, 1979

BOU 80 Bouchenez j.l.,Loyer m., Lucrece l.,Maurice p.,Prusker f.
Sogno j.c.Vercoustre a.m.
Le système Legos, support et système de programmation
Actes des journées BIGRES 1980 ,Rennes

EM2 83 Em2 : un environnement pour Modula2
Falquet g.,Guyot j.,Nerima l. ,université de Genève
Actes des journées Bigres 1983

STO 80 Departement of U.S. Defense
Requirements for Ada Programming Support Environnement
Rapport 'Stoneman' fevrier 1980

FOI 80 conception de programmes assistée par ordinateur
Presentation du projet Sprac
Cert ,rapport final no 3607/deu,cpao/82/1, 1982

KRA 81 Krakowiak s.
Système intégré de production de logiciel : concept et réalisation
Acte des journées Bigres 81 , Grenoble

EST 83 un système automatique de gestion de versions
La base de programmes Adele
Acte des journées Bigres 1983 ,Cap d'Agde

PWB 77 Pwb/unix User's manuel Mai 1977

DEL 81 Procedures Del/Mc
Rapport interne Sems no Del/mc/phm/am/175

LOY 81 Loyer m.
Modularité et compilation séparée : les choix du langage Legos.
Thèse université Paul Sabatier Toulouse. Novembre 81

COQ 81 Pascal/s - spécifications de définition
Document interne Sems no 208704487def00

WIL 81 Willis r.
Aides : Computer Aided Design of Software Systems
Hughes Aircraft Company, Fullerton , California 1981

VER 83 Verniol j.
Compleximètre : outil de mesure de la complexité des logiciels produits dans l'atelier Alis.
Actes des journées Bigres 83, Cap d'Agde.

DOU 82 Doussy J.
Alis : pour une production industrielle du logiciel.
Revue technique Thomson CSF ,Vol 14,No 4 ,décembre 82
Le génie logiciel à la Sems
Actes des journées Bigres ,Grenoble , 1982

SAL 80 Salembier p.
Comparaison des approches prises dans le système de gestion logiciel Galaad et l'environnement de programmation Apse.
Actes des journées Bigres, Rennes 1980

SCH 82 Schmidt
Controlling large software development in a distributed environment projet Cedar
Thesis PHD, University of California , décembre 82, Berkly

SNO 81 Snowdon Ra.
Cades and software developpement, sofware in engineering environments
Proc. Of symposium (S2E2) ,Lahnstein(RFA) North-holland 1981

EDL 81 Edile-s
Spécification de définition
Document interne Sems ref 20871612 DEF 00

DBG 83 Debugger symbolique
Note interne Sems ,Clavaux j.l.