Manuel de Descriptif Informatique
Fascicule D0.03 : Architecture
Document D0.03.01

Architecture générale du *Code_Aster*

Résumé :

On donne dans ce document un aperçu des trois idées qui structurent de façon importante le logiciel *Aster* :

- Le superviseur d'exécution,
- Le gestionnaire de mémoire JEVEUX,
- la notion de calcul élémentaire.

Table des matières

1 Introduction

Le *Code_Aster* est constitué de différents "modules" que l'on peut classer en :

- programme principal FORTRAN 77,
- routines FORTRAN 77 (`subroutine` ou `function`),
- fonctions C,
- routines CAL (CRAY assembling language),
- catalogues.

L'ensemble des textes de ces modules constitue le source d'*Aster* (environ 400 000 lignes). Les catalogues sont des fichiers textes qui **paramètrent** certains programmes : principalement l'analyseur de commandes et les calculs élémentaires (au sens de la méthode des éléments finis).

Les fonctions C réalisent certaines tâches "système" impossibles en FORTRAN 77 : allocation dynamique, mesure de temps, …

Les routines CAL ont été écrites pour optimiser les performances (CPU) de la méthode de résolution des systèmes linéaires par Gradient Conjugué.

Si l'on oublie les quelques fonctions C et les routines CAL, nous voyons que le programme *Aster* est constitué de :

- quelques centaines de catalogues,
- quelques milliers de routines FORTRAN 77.

Le but de ce document est d'aider à se "retrouver" dans ce grand nombre de modules : rien que pour le FORTRAN, nous avons calculé que l'arbre d'appel complet du programme *Code_Aster* s'écrivait sur plus de 6.10^8 lignes, ce qui exclut de le donner en annexe !

Comment, dans ces conditions, identifier les sources relatifs à une fonctionnalité donnée ?
Où insérer une nouvelle fonctionnalité ?

Une forme de réponse à ces 2 questions est dans l'architecture générale du code qui a été choisie au début de Projet (07/89) et qui n'a pas été remise en cause depuis.

Cette architecture peut se résumer à peu près en **trois** idées :

1) *Code_Aster* peut être vu comme un ensemble de commandes indépendantes que l'utilisateur enchaîne à son gré,
2) ces commandes s'échangent des objets nommés ("concepts") qui sont eux mêmes constitués d'objets JEVEUX,
3) *Code_Aster* étant un code d'éléments finis, la notion de calcul "élémentaire" (i.e. fait par un élément fini) est strictement codifiée car elle constitue en quelque sorte le "noyau" de la méthode numérique.

Remarque :

*Les 2 premières idées sont **très générales** et pourraient servir d'architecture à de nombreux logiciels hors du domaine des éléments finis.*

Ce sont ces trois idées que nous allons développer dans les paragraphes suivants :

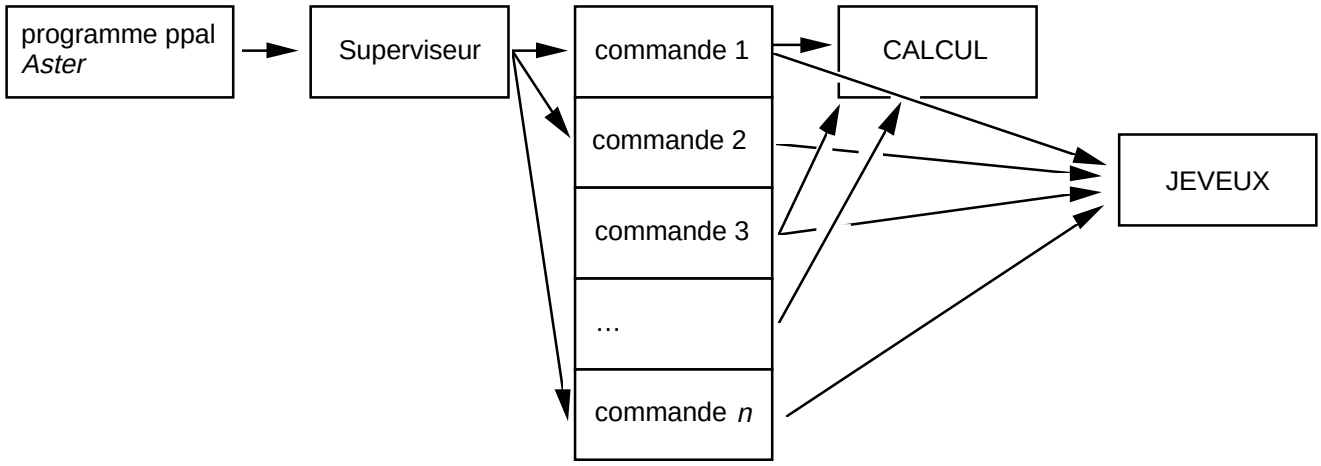
1) s'identifie à peu près à ce que l'on appelle le **Superviseur** d'exécution [§4],
2) est possible grâce au gestionnaire de mémoire **JEVEUX** et à la **Structuration** des **Données** [§5],
3) correspond à la paramétrisation des calculs élémentaires et à l'existence de la routine **CALCUL** [§6].

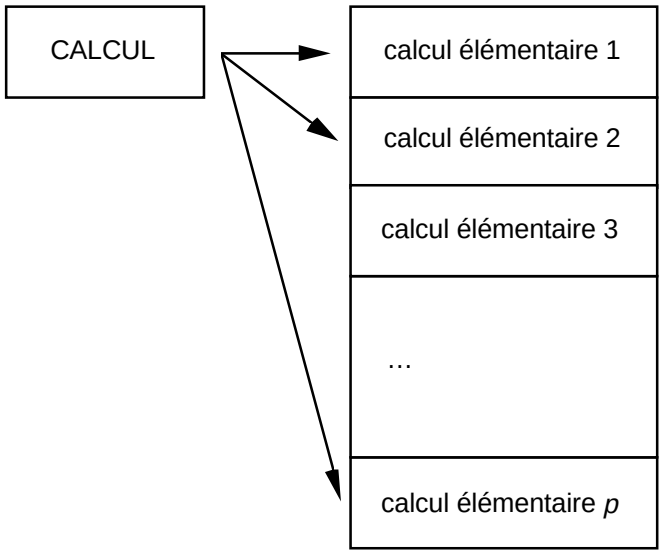
Remarques :

- *l'idée 2 ne structure pas le code à proprement parler, mais elle permet de réaliser 1,*
- *si ces trois idées structurent fortement le code, elles forment également le "carcan" de la programmation : on ne peut s'y soustraire. Le reste de la programmation est à peu près libre,*
- *la mise en œuvre de l'idée 3, outre le fait qu'elle structure un volume important de code (19000 lignes de catalogues et 70000 lignes de FORTRAN), fige un certain nombre de notions essentielles du programme :*
  - *nœud, maille, maillage,*
  - *grandeur, option, élément fini,*
  - *champs, matrices assemblées.*

2 Architecture générale des routines FORTRAN

Schématiquement, l'organisation des routines FORTRAN est la suivante :





Remarque :

Au 01/10/94 : nombre de commandes : n = 128
nombre de calculs élémentaires : p = 3043

Le Superviseur (comme la routine CALCUL) structurent le code car ils **affirment une indépendance** entre les routines qu'ils appellent :

```
routine   0P0001   -->  commande 1
routine   0P0002   -->  commande 2
…         …
routine   TE0001   -->  calcul élémentaire 1
routine   TE0002   -->  calcul élémentaire 2
…         …
```

- le lien entre une commande utilisateur et le numéro i de la routine OP000i qui lui correspond est donné dans le catalogue associé à cette commande [§ 3.1],

- le lien entre un calcul élémentaire (par exemple : le calcul de la rigidité géométrique pour un élément de coque de type DKT) et la routine TE00031 qui lui correspond est donné dans le catalogue associé à cet élément fini [§ 3.2].

L'indépendance entre les routines OP000i est très intéressante. Elle veut dire que pour comprendre le programmation d'une commande on n'a pas besoin de comprendre les autres ; les seuls liens entre les commandes sont les structures de Données qu'elles s'échangent [§ 5.2]. Celles-ci sont décrites dans [D4].

L'indépendance des routines TE000i est plus naturelle (on verra toutefois qu'une même routine TE000i peut être associée à plusieurs calculs élémentaires voisins).

Bien entendu, le schéma précédent ne veut pas dire que tout le source FORTRAN correspondant à la commande i se trouve dans la routine OP000i : le programmeur d'une commande (comme celui d'un calcul élémentaire) peut structurer sa commande comme il l'entend : il peut la "découper" en plusieurs routines.

Schématiquement, on peut écrire :

OP000i --> CALCUL, JEVEUX ou tout autre utilitaire pouvant servir à plusieurs commandes différentes.

--> routines spécifiques à la commande 0P000i (découpage fonctionnel de OP000i)

L'orsque l'on cherche le code source associé à une fonctionnalité donnée, on doit donc se poser les questions suivantes :

- s'agit-il d'une fonctionnalité spécifique à une commande ?
  1) non : voir les utilitaires communs à plusieurs commandes [D7.01],
- s'agit-il d'une fonctionnalité spécifique à un calcul élémentaire ?
  2) non : voir les utilitaires communs à plusieurs calculs élémentaires [D7.02].

3 Architecture des catalogues

Nous distinguons deux sortes de catalogues :

- les catalogues de commandes qui paramètrent le superviseur,
- les catalogues d'éléments qui paramètrent :
  - la routine CALCUL,
  - les commandes LIRE_MAILLAGE et AFFE_MODELE.

3.1 Catalogues de commandes

L'architecture est simple : à chaque commande de nom, `commande_i` correspond un catalogue de même nom. Ces catalogues sont tous indépendants les uns des autres.

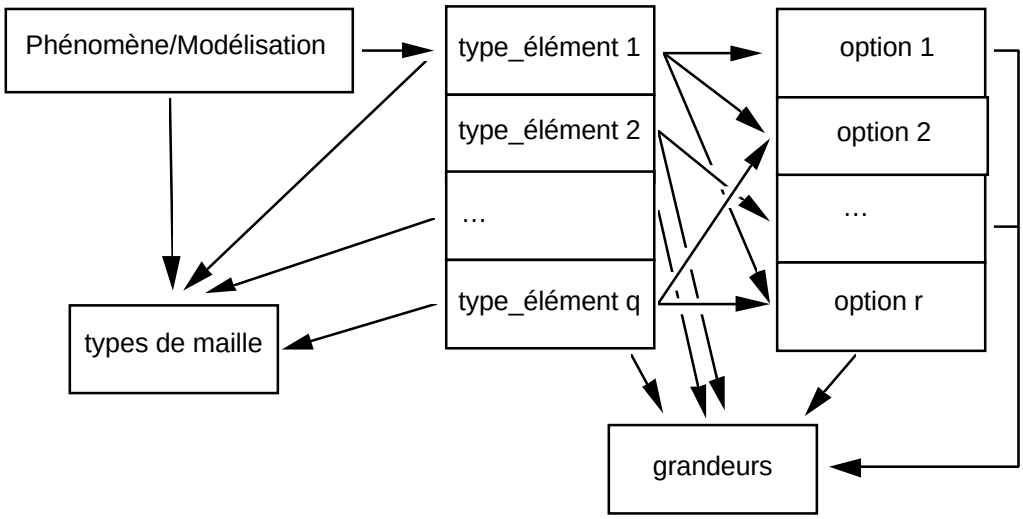
catalogue	`commande_1`
catalogue	`commande_2`
…	
catalogue	`commande_n`

Le contenu du catalogue d'une commande est décrit dans [D5.01.01].

3.2 Catalogues d'éléments finis

L'architecture est là encore assez simple. La description du contenu de ces catalogues est faite dans [D3.02.01].

"cata1 --> cata2" veut dire : le catalogue cata1 s'appuie sur le catalogue cata2. Autrement dit, il utilise des entités décrites dans le catalogue cata2.



Remarque :

Au 01/10/94 :
nombre de type_element : q = 233
nombre d'options : r = 159

4 Le Superviseur d'exécution

4.1 Généralités

Ce que l'on appelle "Superviseur" est l'ensemble des routines FORTRAN qui appartiennent à la bibliothèque SUPERVIS.

On peut le couper en deux logiquement :

SUP1 : ce qui sert à enchaîner les différentes commandes ; c'est-à-dire tout ce qui se trouve (dans l'arbre d'appel) entre le programme principal et les routines OP000i (y compris le programme principal). On inclut dans le superviseur le contenu de 3 commandes particulières : DEBUT, POURSUITE et FIN,

SUP2 : ce qui permet la communication d'informations avec les OP000i : les routines GETXXX [D6.03.01].

4.2 Fonctionnement général du Superviseur

1) ouverture des 3 bases de données JEVEUX [D6.02.01] (en fait 3 fichiers d'accès direct).
   - base "LOCALE" c'est une base de travail réservée au Superviseur, cette base n'est pas sauvegardée en fin d'exécution
   - base "VOLATILE" c'est la base réservée aux objets de travail (hors Superviseur), cette base n'est pas sauvegardée en fin d'exécution,
   - base "GLOBALE" c'est la base de l'utilisateur. Elle contiendra en fin d'exécution les concepts correspondants aux commandes exécutées
2) lecture des catalogues
   - catalogues de commandes
   - catalogues d'éléments
3) lecture du fichier de commandes de l'utilisateur
   - lecture en format libre ; élimination des commentaires,
   - vérifications syntaxiques (à l'aide des catalogues de commandes),
     - orthographe des mots clés,
     - types des arguments,
     - exclusion des mots clés, …
     - affectation des valeurs par défaut,
   - création des concepts correspondants aux valeurs (DEFI_VALEUR [U4.21.10]) et aux fonctions interprétées (FORMULE [U4.21.11]),
   - évaluation des expressions numériques (mot clé EVAL [U4.21.11 §4.1]),
   - stockage des informations du fichier de commandes dans des objets JEVEUX (base locale).
4) demande d'exécution des commandes de l'utilisateur [§ 4.3],
5) impression du temps d'exécution de chaque commande,
6) validation (au fur et à mesure) des concepts créés par les commandes : ceci permet de "reprendre" un calcul qui s'est mal terminé,
7) fermeture des bases de données en fin d'exécution,
8) arrêt du programme.

4.3 Demande d'exécution des commandes

Le Superviseur "boucle" deux fois sur les commandes lues dans le fichier de commandes de l'utilisateur :

1ère passe : phase de vérifications supplémentaires : on vérifie les données de l'utilisateur (ce qui n'a pas pu être vérifié par le superviseur),

2ème passe : phase d'exécution : on exécute véritablement la commande.

Si l'utilisateur a demandé :

`DEBUT (PAR_LOT : 'OUI', …)` (c'est la valeur par défaut)

le Superviseur exécute la 1ère passe sur toutes les commandes avant de commencer la 2ème passe. Ceci permet de vérifier tout le fichier de commandes avant de commencer l'exécution.

Sinon : `DEBUT (PAR_LOT : 'NON', …)`

Le Superviseur exécute les 2 passes l'une après l'autre pour chaque commande.

Remarque :

Le Superviseur enchaîne les commandes les unes après les autres. Les "phrases du langage" (les commandes) se suivent sans instructions de contrôle : IF THEN ELSE, boucles DO,…

4.4 Commandes "Superviseur"

Remarque :

Ce paragraphe peut être sauté en première lecture.

Le paragraphe précédent [§4.3] concernait l'exécution des commandes "ordinaires".

Les commandes ordinaires sont celles dont le numéro est compris entre 1 et 9998.

Les commandes qui ne sont pas ordinaires sont :

- les commandes `DEBUT` et `POURSUITE` qui n'ont pas de catalogue externe,
- la commande `FIN` associée au numéro 9999 qui est chargée (entre autres choses) de décharger la mémoire et de fermer les bases de données,
- les commandes dites "superviseur".

Les commandes Superviseur ont un catalogue (comme les commandes ordinaires), mais leur numéro est un nombre négatif (mot clé `NUMERO_SUPERVIS__` au lieu de `NUMERO__`). Les routines FORTRAN associées se nomment `OPS00i`.

Il existe aujourd'hui (01/10/94) 7 commandes Superviseur.

La différence de comportement entre une commande Superviseur et une commande ordinaire est que le superviseur effectue une passe préliminaire sur les commandes Superviseur. L'idée étant qu'après cette passe préliminaire, tout se passe comme si le fichier de commandes ne contenait que des commandes ordinaires. Cette passe préliminaire peut être considérée comme pré-traitement du fichier de commandes. L'"écho" du fichier de commandes (que l'on retrouve dans le fichier `MESSAGE`) représente l'état du fichier de commandes après cette phase préliminaire.

Les 7 commandes Superviseur actuelles se décomposent en deux : celles qui sont détruites à la fin de la passe préliminaire : `INCLUDE`, `PROC`, `RETURN` et `MACRO_MATR_ASSE` et celles qui ne sont pas

détruites : DEFI_VALEUR, FORMULE et DETRUIRE. Pour ces 3 dernières, on passera donc trois fois dans la routine OPS00i associée : passe préliminaire, passe de vérifications supplémentaires et passe d'exécution.

L'intérêt principal des commandes Superviseur (outre d'avoir permis l'"include", l'emploi des fonctions interprétées et des constantes nommées) est de permettre le développement de "macro" commandes ; MACRO_MATR_ASSE en est un exemple. Lors de la passe préliminaire, la commande MACRO_MATR_ASSE engendre dynamiquement le texte de plusieurs commandes ordinaires puis elle est détruite. Le développement de telles macro-commandes est documenté dans [D5.01.02].

Soit le fichier de commandes :

```
C_O1
C_S1
C_O2
C_S2
C_O3
```

où :

C_Oi sont des commandes ordinaires
C_Si sont des commandes Superviseur

C_S1 est une commande Superviseur de type macro commande qui engendre les commandes ordinaires C_O4 et C_O5.
C_S1 se détruit à la fin de la passe préliminaire

C_S2 est une commande Superviseur qui ne se détruit pas

- Séquence des passes si DEBUT (PAR_LOT : 'oui')
  - passe préliminaire (pour les commandes superviseur seulement) : (pp)
    - C_S1 pp
    - C_S2 pp
  - passe de vérifications supplémentaires : (pv)
    - C_O1 pv
    - C_O4 pv
    - C_O5 pv
    - C_O2 pv
    - C_S2 pv
    - C_O3 pv

- passe d'exécution : (pe)
  - C_01 pe
  - C_04 pe
  - C_05 pe
  - C_02 pe
  - C_S2 pe
  - C_03 pe

- Séquence des passes si DEBUT (PAR_LOT : 'NON')

```
- C_01      pv
- C_01      pe

- C_S1      pp

- C_04      pv
- C_04      pe

- C_05      pv
- C_05      pe

- C_02      pv
- C_02      pe

- C_S2      pp
- C_S2      pv
- C_S2      pe

- C_03      pv
- C_03      pe
```

5 JEVEUX et Structuration des Données

Nous allons dans ce paragraphe tenter de dégager les principales fonctionnalités du gestionnaire de mémoire JEVEUX et de l'usage qu'on en fait dans *Aster*.

5.1 JEVEUX gestionnaire d'objets

JEVEUX est l'ensemble des routines FORTRAN décrites dans [D6.02.01].

Ces routines permettent de :

- créer des objets,
- les sauver (écriture sur disque),
- les détruire,
- les libérer (de la mémoire centrale),
- les rappeler (en mémoire centrale),
- les copier, les imprimer, …

5.1.1 Qu'est-ce qu'un objet JEVEUX ?

- Un ensemble d'informations **homogènes** (entiers, réels, complexes, …),
- chaque objet est nommé (24 caractères),
- chaque objet à des attributs accessibles en lecture (et parfois en écriture) :
  - longueur (pour 1 vecteur),
  - type des valeurs : entier, réel, …
  - …
- chaque objet a virtuellement une "image disque",
- il existe des **objets simples** (grosso modo, ce sont des vecteurs),
- il existe des **collections** d'objets,
  - les objets d'une collection sont tous du même type (mais il peuvent avoir des longueurs différentes),
  - l'accès d'un objet de collection se fait par le nom de la collection plus quelque chose qui identifie l'objet :
    - un numéro (collection numérotée),
    - ou un nom (collection nommée).

5.1.2 Allocation dynamique

On peut créer, libérer (et détruire) à tout instant un objet JEVEUX. Cela permet de gérer dynamiquement la mémoire.

Bien entendu, cette possibilité est utilisée pour allouer des zones de travail. C'est le seul **mécanisme d'allocation dynamique autorisé** dans *Aster* car il gère **l'ensemble** de la place mémoire disponible : (on entend par mémoire disponible la place disponible dans la "Région" demandée à l'exécution moins le volume du code exécutable moins les zones gérées par le système UNICOS).

5.1.3 Mémoire virtuelle

Lorsqu'on libère un objet A, JEVEUX le considère comme "déchargeable" (sur disque). Si de nouvelles requêtes sont faites sur d'autres objets et que la place en mémoire centrale vient à manquer, l'objet A sera écrit sur disque et sa place sera récupérée.

JEVEUX permet donc d'accéder (à des instants différents) à plus de mémoire que n'en contient réellement la "Région" de mémoire centrale allouée à l'exécution.

Il agit donc comme un système de "mémoire virtuelle".

5.1.4 Ecriture et lecture sur disque

- Lorsqu'on sauve explicitement un objet (routine JESAUV), celui-ci est écrit sur disque,
- lorsque l'exécution d'*Aster* se termine, on sauve automatiquement tous les objets de la base globale qui ne l'ont pas encore été,
- lorsqu'on rappelle un objet en mémoire centrale (routine JEVEU0), celui-ci est lu sur le disque s'il a été déchargé et recopié en mémoire centrale.

JEVEUX permet donc de s'affranchir de toutes les lectures et écritures binaires sur disque.

5.2 Structuration des Données

Les commandes d'*Aster* s'échangent des objets nommés par l'utilisateur (8 caractères) que l'on appelle des concepts.

Exemple :
```
acier = DEFI_MATERIAU (ELAS : (E : 300 000 NU : 0.3 ) ) ;
chmat = AFFE_MATERIAU (MATER : acier … ) ;
```

Le concept "acier" créé par la commande DEFI_MATERIAU est un argument d'entrée de la commande AFFE_MATERIAU.

Un concept est en fait une Structure de Données nommée (SD en langage programmeur).

Une structure de donnée n'est rien d'autre qu'un **ensemble** d'objets JEVEUX.

On peut alors "manipuler" dans le FORTRAN des structures de données complexes : le passage de la SD en argument se fait par son nom (chaîne de caractères).

Ceci améliore grandement la définition des interfaces des routines : au lieu de transmettre des multitudes de vecteurs en arguments, on transmet quelques structures de données.

Le regroupement d'un ensemble d'objets JEVEUX dans une structure de données se fait par de simples conventions de noms connues de l'ensemble des programmeurs.

Une Structure de Données est **typée**. Lorsque l'on exécute (par exemple) la commande :

```
mailla = LIRE_MAILLAGE () ;
```

celle-ci doit créer une SD de type maillage et de nom "mailla". A la fin de l'exécution de la commande, il doit exister sur la base 'GLOBALE' un certain nombres d'objets JEVEUX dont l'ensemble forme la SD mailla :

```
'MAILLA  .DIME'
'MAILLA  .CONNEX'
'MAILLA  .NOMNOE'
'MAILLA  .NOMMAI'
…
```

Les 8 premiers caractères des objets sont ceux provenant de l'utilisateur. Les autres caractères (qui servent à distinguer les objets les uns des autres) sont fixés par les programmeurs. La description du contenu des objets .DIME, .CONNEX, … forme ce que l'on appelle la description de la SD de type maillage (cf. [D4]).

Remarque importante :

Les seules informations nécessaires au bon déroulement d'une commande sont :
- *les valeurs que l'utilisateur a fournies derrière les mots clés de la commande : entiers, réels, …*
- *les SD (déjà créées par des commandes précédentes) données en argument.*

Il n'y a pas d'information sous-terraine (COMMONS, fichiers, …) entre les commandes. C'est le respect de cette règle qui assure l'indépendance réelle des commandes entre elles.

Les seules exceptions à cette règle sont :

- la SD catalogue [D4.01.01] qui est accessible partout (mais elle n'est jamais modifiée),
- certaines écritures (ou lectures) dans des fichiers. Dans ce cas, le nom de la commande est toujours de la forme : IMPR_XXX ou (LIRE_XXX).
  Le "format" du fichier peut alors être vu comme la description d'une SD, par exemple :
  - maillage *Aster* (LIRE_MAILLAGE),
  - fonction *Aster* (LIRE_FONCTION),
  - résultats à visualiser par I-DEAS (IMPR_RESU (FORMAT : IDEAS …)).

6 Calculs élémentaires

Nous avons vu au [§2] que les différents calculs élémentaires étaient nombreux dans *Aster*.
Ce nombre important de types de calculs élémentaires résulte :

- du grand nombre d'éléments finis dans les codes de calcul de structures :
  - isoparamétriques 2D en Thermique, Mécanique et Acoustique,
  - isoparamétriques 3D en Thermique, Mécanique et Acoustique,
  - éléments de structures : poutres, coques, ...
  - éléments incompressibles,
  - éléments d'interaction fluide/structure,
  - …
- et du grand nombre d'options de calcul possible :
  - rigidité mécanique ou thermique,
  - masse, amortissement,
  - rigidité géométrique ou centrifuge,
  - contraintes, déformations, flux,
  - forces surfaciques, volumiques ou linéïques,
  - changement de pesanteur, dilatation thermique, …

Dans *Aster* aujourd'hui (01/10/94), on a :

- 233 types d'éléments finis (environ 19000 lignes de catalogues),
- 159 options de calcul élémentaire,

ce qui entraine plus de 3200 calculs élémentaires théoriquement possibles (sans doute beaucoup plus). Seuls 3043 de ces calculs élémentaires sont effectivement programmés (environ 70000 lignes de FORTRAN).

Ces 3043 calculs élémentaires sont faits dans 310 routines TE000i ; en effet, plusieurs calculs élémentaires peuvent être implémentés dans un seul TE000i. Par exemple, il est assez facile de paramétrer la programmation de tous les éléments isoparamétriques 2D.

Le volume important du code concerné par les calculs élémentaires justifie un effort de paramétrisation de ces calculs. Les objectifs de cette paramétrisation sont :

- simplifier au maximum la programmation des TE000i : les données d'un calcul élémentaire "arrivent" dans la routine sous la forme voulue par le programmeur (et décrite dans le catalogue de l'élément [D3.02.01 §7]),
- disposer d'une routine unique (CALCUL) gérant tous les calculs élémentaires : contraintes, rigidité, "masse" thermique, …. Ce qui évite de multiplier les "boucles" sur les éléments, les contrôles et les messages d'erreur : la "fonction" CALCUL représente quand même 3500 lignes ...
- imposer des types de Structures de Données communs à tous les résultats des calculs élémentaires : les CHAM_ELEM (champs par éléments) et les RESU_ELEM (matrices et vecteurs élémentaires).

L'assemblage des matrices et des vecteurs élémentaires peut alors être fait dans deux routines (ASMATR et ASSVEC).

Les mécanismes de cette paramétrisation sont expliqués dans [D3.02.01].

Les documents [D5.04.01] et [D5.04.02] décrivent la manière d'introduire de nouveaux calculs élémentaires.

Page laissée intentionnellement blanche.