General architecture of Code_Aster

Summary:

One gives in this document an outline of the three ideas which structure in an important way the software Aster:

- The supervisor of execution,
- The manager of memory JEVEUX,
- Concept of calculation elementary.

Caution: this document is old and was not updated for a long time.

Contents

1 Introduction

Code_Aster is made up various “modules” which one can classify in:

- main program FORTRANR 77,
- routines FORTRA NR 77 (subroutine or function),
- functions C,
- routines CAL (CRAY assembling language),
- catalogues.

The whole of the texts of these modules constitutes the source of Aster (approximately 400,000 lines). The catalogues are textual files which parameterize certain programs: mainly the analyzer of orders and calculations elementary (within the meaning of the finite element method).

Functions C realize certain impossible tasks “system” in FORTRANR 77 : dynamic allocation, measurement of time,…

Routines CAL were written to optimize the performances (CPU) method of resolution of the linear systems by Combined Gradient.

If the few functions are forgotten C and routines CAL, we see that the program Aster consists of:

- a few hundreds of catalogues,
- a few thousands of routines FORTRYEAR 77.

The goal of this document is to help “to find itself” in this a large number of modules: only for FORTRAN, we calculated that the tree of complete call of the Code_Aster program was written on more than 6,108 lines, which excludes to give it in appendix!

How, under these conditions, to identify the sources relative to a given functionality? Where to insert a new functionality?

A form of answer to these 2 questions is in the general architecture of the code which was selected at the beginning of Project (07/89) and which was not reconsideration since.

This architecture can be summarized about in three ideas:

1. Code_Aster can be seen as a set of independent orders that the user connects with his liking,
2. these orders exchange named objects (“concepts”) who are them same constituted by objects JEVEUX ,
3. Code_Aster being a code of finite elements, the concept of calculation “elementary” (i.e makes by a finite element) is strictly codified because it constitutes to some extent the “core” of the digital method.

Note:

> *The first 2 ideas are **very general** and could be used as architecture with many software out of field of the finite elements.*

They are these three ideas that we will develop in the following paragraphs:

1. be identified about so that one calls it Supervisor of execution (see the § 4 The Supervisor of execution),
2. is possible thanks to the manager of memory JEVEUX and with Structuring Data (see the § 5 JEVEUX and Structuring of the Data),
3. corresponds to the parameterization of elementary calculations and the existence of the routine CALCULATION (see the § 6 Elementary calculations).

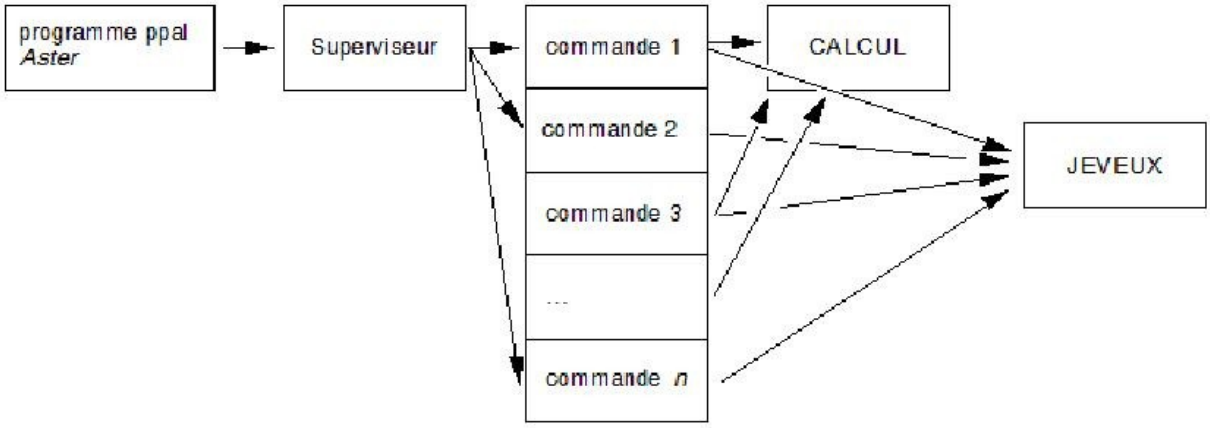
Note:

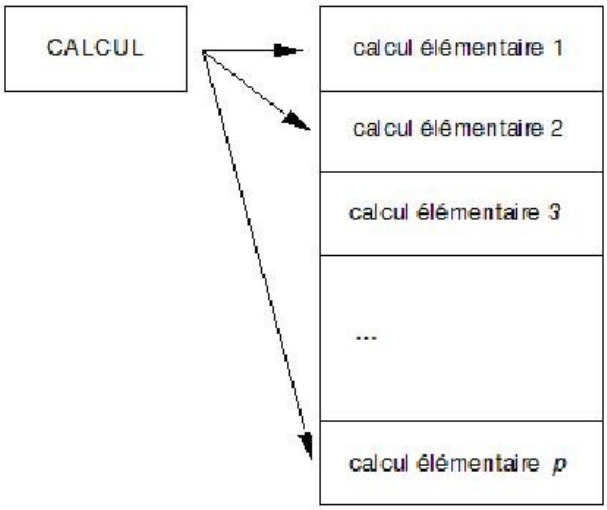
*Idea 2 structure not the code strictly speaking, but it makes it possible to carry out 1,
if these three ideas strongly structure the code, they also form the "yoke" of the programming: one cannot withdraw oneself from it. The rest of the programming is rather free,
the implementation of idea 3, in addition to the fact that it structure a large volume of code (19000 lines of catalogues and 70000 lines of FORTRAN), solidifies a certain number of essential notions of the program:*

→ node, mesh, grid,
→ size, option, finite element,
→ fields, assembled matrices.

2 General architecture of the routines FORTRAN

Schematically, the organization of the routines FORTRAN is the following one:





Note:

*At the 10/1/94:
many orders: N = 128
many elementary calculations: p = 3043*

The Supervisor (as the routine CALCULATION) structure the code because they affirm an independence between the routines which they call:

routine	0P0001	– > order 1
routine	0P0002	– > order 2
…	…	
routine	TE0001	– > elementary calculation 1
routine	TE0002	– > elementary calculation 2
…	…	

- the link between an order user and the number I routine OP000i who corresponds to him is given in the catalogue associated with this order (see the §3.1 Catalogues of orders),
- the link between an elementary calculation (for example: the calculation of geometrical rigidity for an element of hull of the type DKT) and the routine TE0031 who corresponds to him is given in the catalogue associated with this finite element (see the §3.2 Catalogues of finite elements).

Independence between the OP000i routines is very interesting. She wants to say that to understand the programming of order one does not need to understand the others; the only links between the orders are the structures of Data which they exchange (see the § Catalogues Structuring of the Data). Those are described in D4 - Description of the Structures of Data.

The independence of the TE000i routines is more natural (it will be seen however that the same TE000i routine can be associated with several close elementary calculations).

Of course, the preceding diagram does not want to say only all the source FORTRAN corresponding to the order I is in the OP000i routine: the programmer of an order (as that of an elementary calculation) can structure his order as he hears it: he can “cut out it” in several routines.

Schematically, one can write:

OP000i	→	CALCULATION, JEVEUX or any other utility which can be used for several different orders.
	→	routines specific to the order 0P000i (functional cutting of OP000i)

When one seeks the source code associated with a given functionality, one must thus put the following questions:

- is it about a functionality specific to an order?
  not: to see the utilities common to several orders [D7.01],

- is it about a functionality specific to an elementary calculation?
  not: to see the utilities common to several elementary calculations [D7.02].

3 Architecture of the catalogues

We distinguish two kinds of catalogues:

- catalogues of orders which parameterize the supervisor,
- catalogues of elements which parameterize:
  - the routine CALCULATION ,
  - orders LIRE_MAILLAGE and AFFE_MODELE .

3.1 Catalogues of orders

Architecture is simple: to each ordering of name, commande_i corresponds of a the same catalogue name. These catalogues all are independent from/to each other.

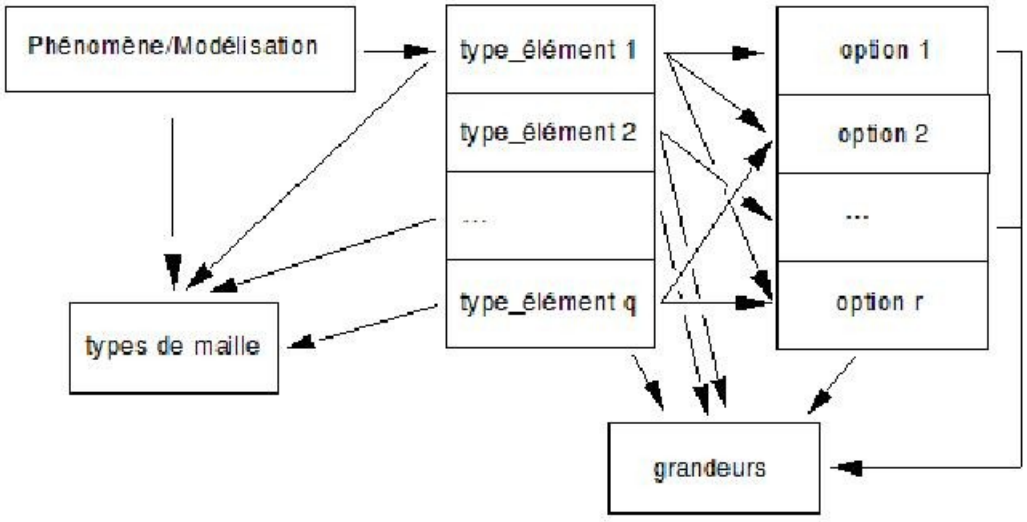
catalogue	commande_1
catalogue	commande_2
…	…
catalogue	commande_n

The contents of the catalogue of an order are described in [D5.01.01].

3.2 Catalogues of finite elements

Architecture is still rather simple there. The description of the contents of these catalogues is made in [D3.02.01].

"cata1 - > cata2" wants to say: the catalogue cata1 be pressed on the catalogue cata2. In other words, it uses entities described in the catalogue cata2.



Note:

At the 10/1/94
number of type_element : Q = 233
many options: R = 159

4 The Supervisor of execution

This paragraph is to be re-examined deeply. That made now a long time that the supervisor was rewritten in Python. On the way, its features were strongly wide.

4.1 General information

What one calls “Supervisor” is the whole of the routines FORTRAN who belong to library SUPERVIS.

One can logically cut it into two:

SUP1:	what is used to connect the various orders; i.e. all that is (in the tree of call) between the main program and the OP000i routines (including the main program). One includes in the supervisor the contents of 3 particular orders: BEGINNING, CONTINUATION and END,
SUP2:	what allows the communication of information with OP000i: routines GETXXX (D6.03.01 - Communication with the Supervisor of execution: routines GETXXX)

4.2 Operation general of the Supervisor

- opening of the 3 databases JEVEUX (D6.02.01 Management memory: JEVEUX) (makes 3 files of direct access of them).
  - base “LOCAL” it is a base of work reserved to the Supervisor, this base is not saved at the end of the execution
  - base “VOLATILE” it is the base reserved for the objects of work (except Supervisor), this base is not saved at the end of the execution,
  - base “TOTAL” it is the base of the user. It will contain at the end of the execution the concepts corresponding to the orders carried out
- reading of the catalogues
  - catalogues of orders
  - catalogues of elements
- reading of the command file of the user
  - reading in free format; elimination of the comments,
  - syntactic checks (using the catalogues of orders),
    - orthography of the keywords,
    - types of the arguments,
    - exclusion of the keywords,…
    - assignment of the values by default,
  - creation of the concepts corresponding to values (DEFI_VALEUR [U4.21.10]) and to the interpreted functions (FORMULA [U4.21.11]),
  - evaluation of the digital expressions (keyword EVAL [U4.21.11 §4.1]),
  - information storage of the command file in objects JEVEUX (bases local).
- request for execution of the orders of the user (Request for execution of the orders),
- impression of the execution time of each order,
- validation (progressively) of the concepts created by the orders: this makes it possible “to take again” a calculation which badly finished,
- closing of the databases at the end of the execution,
- program stop.

4.3 Request for execution of the orders

The Supervisor “buckles” twice on the orders read in the command file of the user:

1era master key:	phase of additional checks: one checks the data of the user (what could not be checked by the supervisor),
2ème master key:	production run: truly the order is carried out.

If the user asked:

BEGINNING (PAR_LOT: 'YES',…) (it is the value by default)

the Supervisor carries out the 1st master key on all the orders before beginning the 2nd master key. This makes it possible to check all the command file before beginning the execution.

If not: BEGINNING (PAR_LOT: 'NOT',…)

The Supervisor carries out the 2 master keys one after the other for each order.

Note:

The Supervisor connects the orders the ones after the others. The "sentences of the language" (orders) are followed without instructions of control: `IF THEN ELSE`, *loops* `C`,...

4.4 Orders "Supervisor"

Note:

This paragraph can be jumped in first reading.

The preceding paragraph (Request for execution of the orders) related to the execution of the "ordinary" orders.

The ordinary orders are those whose number lies between 1 and 9998.

The orders which are not ordinary are:

- orders `BEGINNING` and `CONTINUATION` who do not have an external catalogue,
- the order `END` associated with the number `9999` who is charged (inter alia things) to discharge the memory and to close the databases,
- orders known as "supervisor".

The orders Supervisor have a catalogue (like the ordinary orders), but their number is a negative number (keyword NUMERO_SUPERVIS instead of NUMERO). Routines `FORTRAN` associated name OPS00i.

There exist today (10/1/94) 7 orders Supervisor.

The difference in behavior between an order Supervisor and an ordinary order is that the supervisor carries out a preliminary master key on the orders Supervisor. The idea being that after this preliminary master key, all occurs as if the command file contained only ordinary orders. This preliminary master key can be regarded as preprocessing of the command file. The "echo" of the command file (which one finds in the file MESSAGE) represents the state of the command file after this preliminary phase.

The 7 current orders Supervisor break up into two: those which are destroyed at the end of the preliminary master key: INCLUDE, PROC, RETURN and MACRO_MATR_ASSE and those which are not destroyed: DEFI_VALEUR, FORMULA and TO DESTROY. For these 3 last, one will thus pass three times in the associated OPS00i routine: preliminary master key, passes from additional checks and passes from execution.

The principal interest of the orders Supervisor (in addition to have allowed the "include", use of the interpreted functions and the named constants) is to allow the development of "macro" orders;

MACRO_MATR_ASSE is an example. At the time of the preliminary master key, order MACRO_MATR_ASSE dynamically generates the text of several ordinary orders then it is destroyed. The development of such macro-orders is documented in [D5.01.02].

That is to say the command file:

```
C_O1
C_S1
C_O2
C_S2
C_O3
```

where:

C_Oi	are ordinary orders
C_Si	are orders Supervisor
C_S1	is an order Supervisor of the macro type orders which generates ordinary orders C_O4 and C_O5. C_S1 is destroyed at the end of the preliminary master key
C_S2	is an order Supervisor which is not destroyed

- Sequence of the master keys if `BEGINNING (PAR_LOT: 'YES')`
  - preliminary master key (for the orders supervisor only): (`pp`)

- C_S1	pp
- C_S2	pp

- master key of additional checks: (`pv`)

- C_O1	pv
- C_O4	pv
- C_O5	pv
- C_O2	pv
- C_S2	pv
- C_O3	pv

- master key of execution: (`EP`)

- C_O1	EP
- C_O4	EP
- C_O5	EP
- C_O2	EP
- C_S2	EP
- C_O3	EP

- Sequence of the master keys if `BEGINNING (PAR_LOT: 'NOT')`

```
- C_O1 pv
- C_O1 EP

- C_S1 pp

- C_O4 pv
- C_O4 EP
```

```
- C_O5 pv
- C_O5 EP

- C_O2 pv
- C_O2 EP

- C_S2 pp
- C_S2 pv
- C_S2 EP

- C_O3 pv
- C_O3 EP
```

5 JEVEUX and Structuring of the Data

We will try in this paragraph to release the principal features of the manager of memory JEVEUX and of the use that one makes some in Aster.

5.1 JEVEUX manager of objects

JEVEUX is the whole of the routines FORTRAN described in D6.02.01 Management memory: JEVEUX.

These routines allow:

- to create objects,
- to save (writing on disc),
- to destroy,
- to release (main memory),
- to recall (in main memory),
- to copy, print them,…

5.1.1 What an object JEVEUX ?

- A set of information homogeneous (entireties, realities, complexes,…),
- each object is named (24 characters),
- each object with accessible attributes in reading (and sometimes in writing):
  - length (for 1 vector),
  - type of the values: entirety, reality,…
  - …
- each object has a “image disc virtually”,
- there exists simple objects (roughly speaking, they are vectors),
- there exists collections objects,
  - the objects of a collection are all of the same type (but it can have different lengths),
  - the access of an object of collection is done by the name of the collection plus something which identifies the object:
    - a number (numbered collection),
    - or a name (named collection).

5.1.2 Dynamic allocation

One can create, at any moment to release (and destroy) an object JEVEUX. That makes it possible to manage the memory dynamically.

Of course, this possibility is used to allocate working areas. It is the only mechanism of dynamic allocation authorized in Aster because it manages the whole of the place memory available: (one understands by memory available the place available in the “Area” requested from the execution minus the volume of the achievable code minus the zones managed by system UNICOS).

5.1.3 Virtual memory

When an object A is released, JEVEUX regards as “déchargeable” (on disc). So of new requests are made on other objects and that the place in main memory has suddenly missed, object A will be written on disc and its place will be recovered.

JEVEUX thus allows to reach (at different moments) more memory than really the “Area” of main memory allocated with the execution does not contain any.

It thus acts like a system of “virtual memory”.

5.1.4 Writing and reading on disc

- When one saves an object (routine explicitly JESAUV), this one is written on disc,
- when the execution of Aster finishes, one automatically saves all the objects of the total base which were not it yet,
- when one points out an object in main memory (routine JEVEUO), this one is read on the disc if it were discharged and recopied in main memory.

JEVEUX thus allows to free itself from all the binary readings and writings on disc.

5.2 Structuring of the Data

Orders of*Aster* objects named by the user (8 characters) are exchanged whom one calls of the concepts.

Example:

```
steel = DEFI_MATERIAU (ELAS: (E: 300,000 NAKED: 0.3));
chmat = AFFE_MATERIAU (MATER: steel…);
```

The concept “steel” created by order DEFI_MATERIAU is an argument of entry of order AFFE_MATERIAU.

A concept is in fact a named Structure of Data (SD in language programmer).

A structure of data is anything else only a set of objects JEVEUX.

One can then “handle” in FORTRAN structures of data complex: the passage of the SD in argument is done by its name (character string).

This largely improves the definition of the interfaces of the routines: instead of transmitting multitudes of vectors in arguments, one transmits some structures of data.

The regrouping of a set of objects JEVEUX in a structure of data is done by simple known conventions of names of the whole of the programmers.

A Structure of Data is typified. When one carries out (for example) the order:

```
= LIRE_MAILLAGE netted ();
```

This one must create a SD of type grid and of name “netted”. At the end of the execution of the order, it must exist on the ‘TOTAL’ basis certain numbers of objects JEVEUX whose whole forms the SD netted:

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

The first 8 characters of the objects are those coming from the user. The other characters (which are used to distinguish the objects from/to each other) are fixed by the programmers. The description of the contents of objects .DIME, .CONNEX,… form what one calls the description of the SD of type grid (cf D4 - Description of the Structures of Data).

Notice important:

The only necessary information with the good progress of an order are:
→ the values which the user provided behind the keywords of the order: entireties, realities,…
→ SD (already created by preceding orders) given in argument.

There is no information under-terraine (COMMONS , files,…) between the orders. It is the compliance with this rule which ensures the real independence of the orders between them.

The only exceptions to this rule are:

- the SD catalogues [D4.01.01] which is accessible everywhere (but it is never modified),
- certain writings (or readings) in files. In this case, the name of the order is always of the form:
  IMPR_XXXX or (LIRE_XXXX).
  The “format” of the file can then be seen like the description of a SD, for example:
  - grid Aster (LIRE_MAILLAGE),
  - function Aster (LIRE_FONCTION),
  - results to visualize by I-DEAS (IMPR_RESU (FORMAT: IDEAS…)) .

6 Elementary calculations

We saw with the §2 (General Architecture of the routines FORTRAN) that various elementary calculations were numerous in Aster

This significant number of the elementary types of calculations results:

- a large number of finite elements in the computer codes of structures:
  - isoparametric 2D in Thermics, Mechanics and Acoustics,
  - isoparametric 3D in Thermics, Mechanics and Acoustics,
  - elements of structures: beams, hulls,…
  - incompressible elements,
  - elements of fluid interaction/structure,
  - …
- and of a large number of options of possible calculation:
  - mechanical or thermal rigidity,
  - mass, damping,
  - geometrical rigidity or centrifuges,
  - constraints, deformations, flow,
  - surface, voluminal or linear forces,
  - change of gravity, thermal dilation,…

In Aster today (10/1/94), one a:

- 233 types of finite elements (approximately 19000 lines of catalogues),
- 159 options of elementary calculation,

what involves more than 3200 theoretically possible elementary calculations (undoubtedly much more). Only 3043 of these elementary calculations are actually programmed (approximately 70000 lines of FORTRAN).

These 3043 elementary calculations are done in 310 TE000i routines; indeed, several elementary calculations can be implemented in only one TE000i. For example, it is rather easy to parameterize the programming of all the isoparametric elements 2D.

The large volume of the code concerned with elementary calculations justifies an effort of parameterization of these calculations. The objectives of this parameterization are:

- to simplify to the maximum the programming of TE000i : the data of an elementary calculation "arrive" in the routine in the form wanted by the programmer (and described in the catalogue of the element [D3.02.01 §7]),
- to have a single routine (CALCULATION) managing all elementary calculations: constraints, rigidity, thermal "mass",…. What avoids multiplying the "loops" on elements, controls and error messages: the "function" CALCULATION represent 3500 lines nevertheless…
- to impose types of Structures of Data commun runs on all the results of elementary calculations: CHAM_ELEM (fields by elements) and them RESU_ELEM (elementary matrices and vectors).

The assembly of the matrices and the elementary vectors can then be made in two routines (ASMATR and ASSVEC).

The mechanisms of this parameterization are explained in [D3.02.01].

The documents [D5.04.01] and [D5.04.02] describe the manner of introducing new elementary calculations.