General architecture of the Summarized

Code_Aster:

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 JEVEUX memory ,
- notion of elementary computation.

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

Do contents

1 Introduction..... 3
2 general Architecture of routines FORTRAN..... 4
3 Architecture of the catalogs..... 5.3.1
Catalogs of commands..... 6.3.2
Catalogs of finite elements..... 6
4 the Supervisor of execution..... 7.4.1
General information..... 7.4.2
general Operation of Supervisor..... 7.4.3
Require execution of the commands..... 7.4.4
Commands “Supervisor”..... 8
5 JEVEUX and Structuring of the Data..... 10.5.1
JEVEUX manager of objects.....
10.5.1.1 What a JEVEUX object?.....
10.5.1.2 Dynamic allocation.....
10.5.1.3 virtual Memory.....
11.5.1.4 Writing and reading on disc..... 11.5.2
Structuring of the Data..... 11
6 Elementary computations..... 12

1 Introduction

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

program main FORTRAN 77,
routines FORTRA N 77 (subroutine or function),
functions C,
routines CAL (CRAY assembling language),
catalogs.

All the texts of these moduli constitute the source of Aster (approximately 400.000 lines). The catalogs are textual files which parameterize certain programs: mainly the analyzer of commands and the elementary computations (within the meaning of the finite element method).

The functions C carry out certain impossible tasks “system” in FORTRAN 77 : dynamic allocation, measurement of time,…

routines CAL were written to optimize performances (CPU) of the method of resolution of the linear systems by Conjugate gradient.

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

a few hundreds of catalogs,
a few thousands of routines FORTRAN 77.

The goal of this document is to help “to find itself” in this a large number of moduli: only for FORTRAN, we calculated that the shaft of complete call of the program Code_Aster 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 response 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 commands that the user connects with his liking,
2. these commands exchange named objects (“concepts”) which are them same constituted by JEVEUX objects ,
3. Code_Aster being a code of finite elements, notion of “elementary” computation (i.e makes by a finite element) is strictly codified because it constitutes to some extent the “core” of the numerical 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:

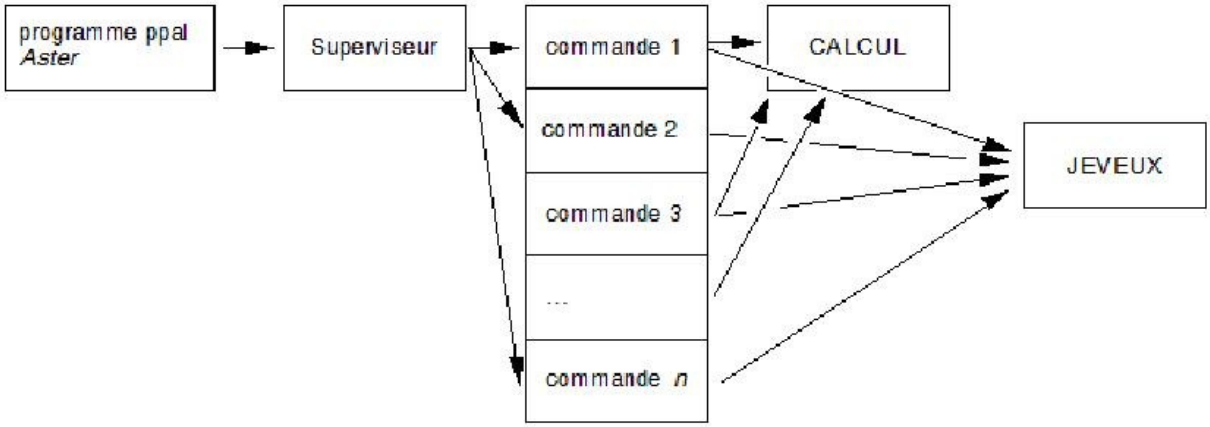
1. be identified about so that one calls the Supervisor of execution (see the § 4 4 the Supervisor of execution), is
2. possible thanks to the manager of JEVEUX memory and the Structuring of the Data (see the § 5 5 and Structuring of the Data), corresponds
3. to the parameterization of elementary computations and the existence of the routine CALCUL (see the § 6 6). Note:

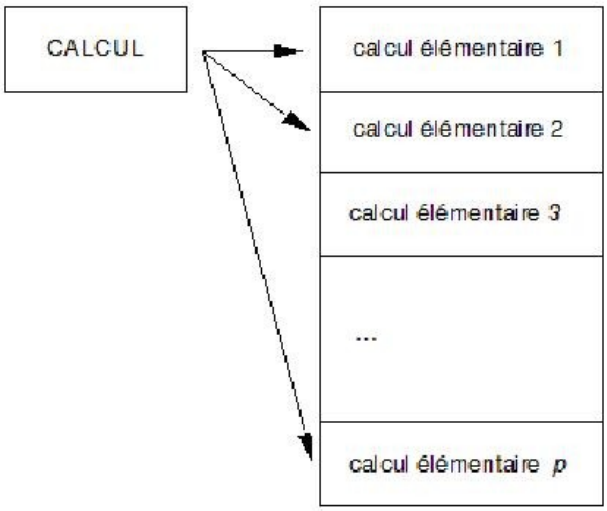
Idea 2 structure not the code strictly speaking, but it makes it possible to carry out 1, if these three ideas strongly structure the code, they also form the "yoke" of the programming: one cannot withdraw oneself from it. The rest of the programming is rather free, the implementation of idea 3, in addition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

node, mesh, mesh, →
quantity, option, finite element, →
fields, assembled matrixes. General

2 architecture of routines FORTRAN Schematically

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





At

the 10/1/94: many commands: N = 128 many elementary computations: p = 3043

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

0P	0001 –	> command 1 routine
0P	0002 –	> command 2…
…	routine	
YOU	0001 –	> elementary computation 1 routine
YOU	0002 –	> elementary computation 2…
…		

the restrain between a command user and number I of the routine OP 000i who corresponds to him is given in the catalog associated with this command (see the §3.13.1 of commands),

the restrain between an elementary computation (for example: the computation of the geometrical stiffness for a shell element of the type DKT) and the routine YOU 0031 who corresponds to him is given in the catalog associated with this finite element (see the §3.23.2 of finite elements).

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

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 computations). Of course

, the preceding diagram does not want to say that all source FORTRAN corresponding to command I am in the OP000i routine: the programmer of a command (as that of an elementary computation) can structure his command as he hears it: he can “cut out it” in several routines. Schematically

, one can write: OP

000i →	CALCUL	, JEVEUX or any other utility which can be used for several different commands. →
	routines	specific to the command 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 a command? not
: to see does the utilities common to several commands [D7.01],

act of a functionality specific to an elementary computation? not
: to see the utilities common to several elementary computations [D7.02]. Structure

3 of the catalogs We

distinguish two kinds of catalogs:

catalogs of commands which parameterize the supervisor,
catalogs of elements which parameterize:
the routine CALCUL ,
commands LIRE_MAILLAGE and AFFE_MODELE . Catalogs

3.1 of commands

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

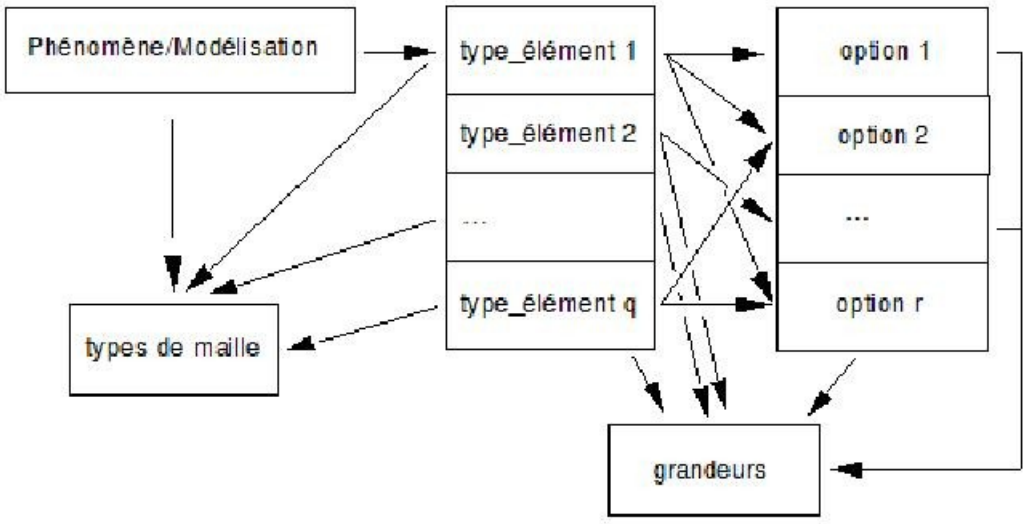
command	_1 catalog
orders	_2…
…	catalogues
command	_n

the contents of the catalog of a command is described in [D5.01.01]. Catalogs

3.2 of finite elements

architecture is still rather simple there. The description of the contents of these catalogs is made in [D3.02.01]. “

catastrophes1 - > cata2” wants to say: the catalog catastrophes 1 leans on the catalog catastrophes 2. In other words, it uses entities described in the catalog catastrophes 2. Note:



At

the 10/1/94 number
of the type _element: Q = 233 many
options: R = 159

4 the Supervisor of execution This

paragraph are 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. General information

4.1

what one calls “Supervisor” is all the routines FORTRAN which belong to library SUPERVIS. One

can logically cut it into two: SUP

1: what	is used to connect the various commands; i.e. all that is (in the shaft of call) between the main program and the OP000i routines (including the main program). One includes in the supervisor the contents of 3 particular commands: debut, POURSUITE and FIN, SUP
2: what	allows the communication of information with OP000i: routines GETXXX (D6.03.01 - Communication with the Supervisor of execution: routines GETXXX) general

4.2 Operation of the Supervisor opening

- of 3 data bases JEVEUX (D 6.02.01 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 bases
  - “VOLATILE” it is the base reserved for the objects of work (except Supervisor), this base is not saved at the end of the execution, bases
  - “TOTAL” it is the base of the user. It will contain at the end of the execution the concepts corresponding to the commands carried out reading
- of the catalogs catalogs
  - of commands catalogs
  - of elements reading
- of the command file of the user reading
  - in free format; elimination of the comments, syntactic
  - checks (using the catalogs of commands), orthography
    - of the key, standard keys
    - of the arguments, exclusion
    - of the key keys,… assignment
    - of the default values, creation
  - of the concepts corresponding to values (DEFI_VALEUR [U4.21.10]) and to the interpreted functions (FORMULA [U4.21.11]), evaluating
  - of the numerical statements (key word EVAL [U4.21.11 §4.1]), information storage
  - of the command file in JEVEUX objects (bases local). ask
- execution of the commands of the user (Request for execution of the commands), printing
- of the execution time of each command, validation
- (progressively) of the concepts created by the commands: this makes it possible “to take again” a computation which badly finished, closing
- of data bases at the end of the execution, program stop
- . Ask

4.3 to execution of the commands

the Supervisor “buckles” twice on the commands read in the command file of the user: 1st

master key : phase	of additional checks: one checks the data of the user (what could not be checked by the supervisor), 2nd
master key : stage of execution	: truly the command is carried out. If

the user asked: DEBUT

(PAR_LOT: “YES”,…) (it is the value by default)

the Supervisor carries out the 1st master key on all the commands before beginning the 2nd master key. This makes it possible to check all the command file before beginning the execution. If not

: DEBUT (PAR_LOT: “NON”,…)

The Supervisor carries out the 2 master keys one after the other for each command. Note:

The Supervisor connects the commands the ones after the others. The “sentences of the language” (commands) are followed without instructions of control: IF THEN ELSE, loops C ,… Commands

4.4 “Supervisor” Note

: This
paragraph can be jumped in first reading.

The preceding paragraph (Request for execution of the commands) related to the execution of the “ordinary” commands.

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

The commands which are not ordinary are:

the commands debuts and POURSUITE which do not have an external catalog,
command FIN associated with the number 9999 which is charged (inter alia things) to discharge the memory and to close data bases,
commands known as “supervisor”.

The commands Supervisor have a catalog (like the ordinary commands), but their number is a negative number (key word NUMERO_SUPERVIS instead of NUMERO). Associated routines FORTRAN name OPS00i. There

exist today (10/1/94) 7 commands Supervisor.

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

The 7 current commands 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: passes preliminary, passes from additional checks and passes from execution.

The principal interest of the commands Supervisor (in addition to have allowed the “include”, use of the interpreted functions and the named constants) is to allow the development of “macro” commands; MACRO_MATR_ASSE is an example. At the time of the preliminary master key, command MACRO_MATR_ASSE dynamically generates the text of several ordinary commands then it is destroyed. The development of such macro-commands 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	C_ ordinary commands
If are	commands Supervisor C_
S1 is	a command Supervisor of type macro command which generates ordinary commands C_O4 and C_O5. C_ S1 is destroyed at the end of the preliminary master key C_
S2 is	a command Supervisor which does not destroy Sequence

of the master keys if debut (PAR_LOT: "YES") places
preliminary (for the orders supervisor only): (pp) -

C_S1 pp	-
C_S2 pp	passes

from additional checks: (pv) -

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

from 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 debut (PAR_LOT: "NON") -

```
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 JEVEUX
```

5 and Structuring of the Data We

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

5.1 manager of JEVEUX objects

is all routines FORTRAN described in D6.02.01 Management memory: JEVEUX . These

routines allow: to create

objects, to save
them (writing on disc), to destroy
them, release
them (main memory), to point out
them (in main memory), to copy
them, print them,… What

5.1.1 a JEVEUX object ? A set of

homogeneous information (integers , 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: integer, reality,……
each

object virtually has a “disk image”, it
exists simple objects (roughly speaking , they are vectors), it
exists collections of 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). Dynamic allocation

5.1.2 One

can create, to release (and destroy) at any moment a JEVEUX object . 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 group of the core memory available: (one understands by memory available core available in the “Region” requested from the execution minus the volume of the executable code minus the zones managed by system UNICOS). Virtual

5.1.3 memory Lorsqu

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

thus makes it possible to reach (at different times) 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”. Writing

5.1.4 and reading on Lorsqu disc

“one saves explicitly an object (routine JESAUV), this one is written on disc, when
L” execution of Aster finishes, one automatically saves all the objects of the global database which were not it yet, lorsqu
“one points out an object in main memory (routine JEVEUO), this one is read on the disc S” it was discharged and recopied in main memory. JEVEUX

thus makes it possible to be freed from all the binary readings and writings on disc. Structuring

5.2 of the Data

the commands of Aster exchange objects named by the user (8 characters) whom one calls of the concepts. Example:

steel

```
= DEFI_MATERIAU (ELAS: (E: 300.000 NU: 0.3)); chmat
= AFFE_MATERIAU (MATER: steel…);
```

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

A concept in fact one is Data format named (SD in language programmer).

A data structure is nothing other than a set of JEVEUX objects . One

can then “handle” in FORTRAN of complex data structures: the transition 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 data structures.

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

Data format is typified. When one carries out (for example) the command: =

```
LIRE_MAILLAGE netted (); This one
```

must create a SD of mesh type and of name “netted”. At the end of the execution of the command, it must exist on the “TOTAL” basis certain numbers of JEVEUX objects whose group forms the SD netted: “.DIME

```
NETTED  ” “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 of mesh type calls the description of the SD (cf D4 - Description of Data structures). Notice

important:

The only necessary information with the good progress of a command are: →
the values which the user provided behind the keywords of the command: integers, realities,… →
SD (already created by preceding commands) data in argument. There

is no information under-terraine (COMMONS , files,…) between the commands. It is the compliance with this rule which ensures the real independence of the commands 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 command is always of the form:
IMPR or (TO READ) .
The “format” of the file can then be seen like the description of a SD, for example: Aster mesh
(LIRE_MAILLAGE), function
Aster (TO READ _FONCTION), results
to visualize by I-DEAS (IMPR_RESU (FORMAT: IDEAS…)) . Elementary computations

6 We

saw with the §2 (2 Architecture of routines FORTRAN) that various elementary computations were numerous in Aster This

significant number of elementary types of computation results:

a large number of finite elements in the structure computer codes: isoparametric
2D in Thermal, Mechanics and Acoustics, isoparametric
3D in Thermal, Mechanics and Acoustics, structural elements

: beams, shells,… incompressible
elements, elements
of fluid interaction/structure,…
and
of a large number of computation options possible: stiffness
mechanical or thermal, mass
, damping, geometrical
stiffness or centrifuge, stresses
, strains, flux, forces
surface, voluminal or linear, change
of gravity, thermal thermal expansion,… In

Aster (10/1/94), one a: 233 element types

finished today (approximately 19000 lines of catalogs), 159
computation options elementary, which

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

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

The large volume of the code concerned with elementary computations justifies a force of parameterization of these computations. The purposes of this parameterization are: to simplify

- to the maximum the programming of TE 000i: the data of an elementary computation “arrive” in the routine in the form wanted by the programmer (and described in the catalog of the element [D3.02.01 §7]), to have
- a single routine (CALCUL) managing all elementary computations: stresses, stiffness, thermal “mass”,…. What avoids multiplying the “loops” on the elements , controls and the error messages: the “function” CALCUL represents 3500 lines nevertheless… to impose
- types of Data structures common to all the results of elementary computations: the CHAM_ELEM (fields by elements) and the RESU _ELEM (elementary matrixes and vectors).

The assembly of the matrixes 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 way introduce new elementary computations.