P2−33

The CAOS Application Builder

Luca Fini, Marcel Carbillet, Armando Riccardi
Osservatorio Astrofisico di Arcetri. Firenze. Italy
lfini@arcetri.astro.it

Introduction

The CAOS software system is a set of tools specifically designed to allow the modeling of any kind of Adaptive Optics systems originally developed in the framework of the "TMR Network on Laser guide star for 8 meter class telescopes" funded by the European Community. In this same conference an overall description of the original package (see: [P2−32]) and a specialized version dedicated to the deconvolution of multiple interferometric images for the Large Binocular Telescope (see: [P1−12]) are also presented.

The "Application Builder" has been designed in order to provide the scientists with a Graphical Programming Environment in which elementary building blocks could be assembled together to create complex simulation applications in a straightforward manner, so that the user can concentrate on the scientific aspects of his/her problem, while mundane coding problems are managed by some automatic tool.

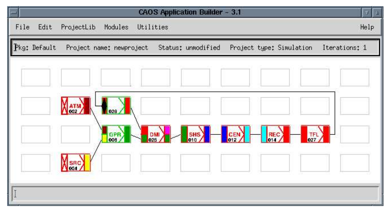
The functionalities and the overall architecture of the AB are the result of a tradeoff among various requirements, and mainly of three principal goals:

- The programming effort for the development of the AB had to be small; i.e.: much lower than the programming effort devoted to the development of the full package.
- The AB must have marginal impact on the structure of simulation programs; i.e.:: the coding of building blocks must not be affected too much and the time efficiency at run−time of the code produced must not be decreased significantly.
- The requirements on the coding of building blocks imposed by the use of the AB must not prevent the use of single blocks in the traditional way, i.e.: as usual routines to be called by a user written program.

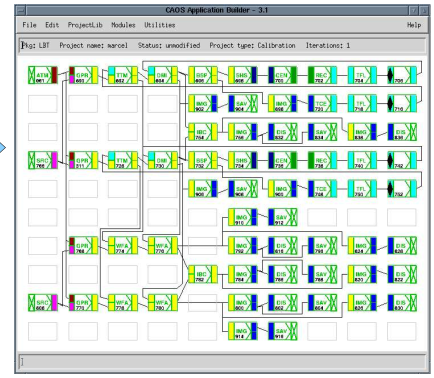
How the Application Builder Works

When the Application Builder is launched it appears to the user as a graph−ical window (the worksheet) provided with a number of rectangular slots and with a number of pull−down menus.

The building blocks, called modules can be selected from a list and placed on the worksheet to build up the simulation program (the project); modules can be put into any free slot on the worksheet and then inputs and outputs can be joined by means of links which represent the data flow in the program.



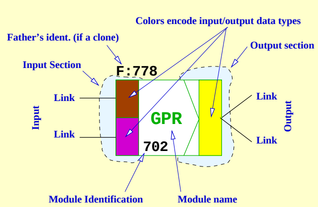
A project is being developed in the Application Builder



A more complex project

The Modules

Modules are represented as "computational blocks" provided with up to two inputs and up to two outputs. In order to convey the concept of input and output data types, the input and output types are encoded by different colors, so that it is clear that only equally colored inputs and outputs can be joined together.

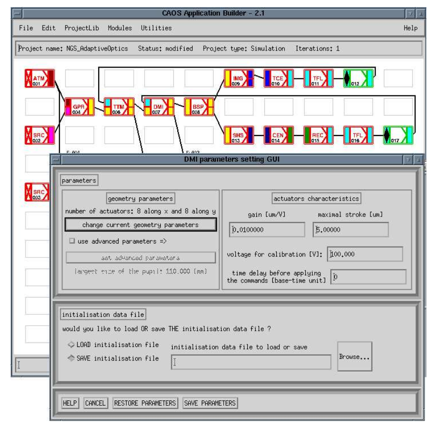


A few modules have been designed to be "generic−type", i.e.: they accept in−put (output) of any type and the actual type is assigned when they are linked to some typed output (input). This mechanism is useful for general purpose modules (e.g.: the data display module) as an alternative to providing a par−ticular typed module to display data values of each particular data type. The generic data display module has been implemented so that it checks the data type of its input and uses the appropriate section of code to display the value of a particular item.

For any module which requires the specification of run−time parameters the user can fire a Graphical User Interface which helps in the definition of values to be used in the computation of the program. At exit the parameter definition GUI creates the required data structures and saves them onto the working directory by means of and IDL standard save command.

It often happens, in the design of a simulation program, that the same module must be used twice or more times in different parts of the program. E.G.: this is the case of an optical element which is used twice in the optical path. This case is handled by the AB by means of "module clones": all modules of the same type share the implementation code, clones also share the set of run−time parameters. When the parameter definition GUI is started for a clone, the parameters which are actually modified are those of the "father" module. This ensures that the two modules correctly represent the same physical ele−ment.

A special "feedback" module must be used to "close loops", i.e.: to assemble simulation programs which include feedback loops.



The Graphical Parameter Definition Interface for the DMI Module

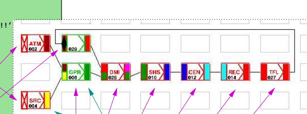
Code Generation

When the project is finished it can be saved on disk. The save operation generates the source code which implements the simulation program together with a textual description of the graphic layout of the project by which the project can be later restored in the Application Builder for subsequent use.

The code generated consists of two IDL procedures: mod_calls.pro, which contains the sequence of procedure calls corresponding to the project, and project.pro, containing initialization and looping instructions.

File: mod_calls.pro

```
COMMON caos_block, tot_iter, this_iter,   $
       calibration, signature

IF signature NE    972657937 THEN BEGIN
        PRINT, 'PROCEDURE FILES NOT ALIGNED!!'
        RETURN
ENDIF

ret = src(O_004_00,             $
          src_00004_p,          $
          INIT=src_00004_c)
IF ret NE 0 THEN ProjectMsg, "src"

ret = atm(O_002_00,             $
          atm_00002_p,          $
          INIT=atm_00002_c)
IF ret NE 0 THEN ProjectMsg, "atm"

;-------------------------------- Loop is closed Here
IF N_ELEMENTS(O_027_00) GT 0 THEN O_029_00 = O_027_00
;----------------------------------------------------

ret = gpr(O_004_00,             $
          O_002_00,             $
          O_006_00,             $
          gpr_00006_p,          $
          INIT=gpr_00006_c)
IF ret NE 0 THEN ProjectMsg, "gpr"

ret = dmi(O_006_00,             $
          O_029_00,             $
          O_025_00,             $
          O_025_01,             $
          dmi_00025_p,          $
          INIT=dmi_00025_c,     $
          TIME=dmi_00025_t)
IF ret NE 0 THEN ProjectMsg, "dmi"

ret = shs(O_025_00,             $
          O_010_00,             $
          shs_00010_p,          $
          INIT=shs_00010_c,     $
          TIME=shs_00010_t)
IF ret NE 0 THEN ProjectMsg, "shs"

ret = cen(O_010_00,             $
          O_012_00,             $
          cen_00012_p,          $
          INIT=cen_00012_c)
IF ret NE 0 THEN ProjectMsg, "cen"

ret = rec(O_012_00,             $
          O_014_00,             $
          rec_00014_p,          $
          INIT=rec_00014_c)
IF ret NE 0 THEN ProjectMsg, "rec"

ret = tfl(O_014_00,             $
          O_027_00,             $
          tfl_00027_p,          $
          INIT=tfl_00027_c)
IF ret NE 0 THEN ProjectMsg, "tfl"
```



File: project.pro

```
COMMON caos_block, tot_iter, this_iter, $
       calibration, signature

tot_iter =     1000
signature =    972657937
this_iter = 0
calibration =        0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Load Parameter variables ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; - Module: src_00004
src_00004_c=0
RESTORE, 'Projects/luca1/src_00004.sav'
src_00004_p=par

; - Module: atm_00002
atm_00002_c=0
RESTORE, 'Projects/luca1/atm_00002.sav'
atm_00002_p=par

; - Module: gpr_00006
gpr_00006_c=0
RESTORE, 'Projects/luca1/gpr_00006.sav'
gpr_00006_p=par

; - Module: dmi_00025
dmi_00025_c=0
dmi_00025_t=0
RESTORE, 'Projects/luca1/dmi_00025.sav'
dmi_00025_p=par

; - Module: shs_00010
shs_00010_c=0
shs_00010_t=0
RESTORE, 'Projects/luca1/shs_00010.sav'
shs_00010_p=par

; - Module: cen_00012
cen_00012_c=0
RESTORE, 'Projects/luca1/cen_00012.sav'
cen_00012_p=par
```

```
; - Module: cen_00012
cen_00012_c=0
RESTORE, 'Projects/luca1/cen_00012.sav'
cen_00012_p=par

; - Module: rec_00014
rec_00014_c=0
RESTORE, 'Projects/luca1/rec_00014.sav'
rec_00014_p=par

; - Module: tfl_00027
tfl_00027_c=0
RESTORE, 'Projects/luca1/tfl_00027.sav'
tfl_00027_p=par

;;;;;;;;;;;;;;;;;
; Initialization;
;;;;;;;;;;;;;;;;;

print, "=== INITIALIZATION... ==="
@Projects/luca1/mod_calls.pro

;;;;;;;;;;;;;;;;
; Loop Control ;
;;;;;;;;;;;;;;;;

print, "=== RUNNING... ==="
FOR this_iter=1, tot_iter DO BEGIN
        @Projects/luca1/mod_calls.pro
ENDFOR

;;;;;;;;;;;;;;;;;
; End Main      ;
;;;;;;;;;;;;;;;;;

END
```

On top of file project.pro a set of lines are added to describe the graphical details of the project. These lines are syntactically IDL comments and are thus irrelevant for the simulation program.

As an example here follows the graphical description of a small part of the project to the left:

```
     ....
; MODULE: gpr
;       6 - ID
;       1 - STATUS
;       1       1 - SLOT
;       2 - Ninputs
;       4       0 - Input       0    Nptns:           2   Dtype:       6
;               70     385
;               90     427
;       2       0 - Input       1    Nptns:           2   Dtype:       1
;               70     485
;               90     442
;       1 - Noutputs
;       0 - Output   Dtype:       7
; MODULE: src
;       4 - ID
;       0 - STATUS
;       0       2 - SLOT
;       0 - Ninputs
;       1 - Noutputs
;       0 - Output   Dtype:       6
   ....
```

References

Fini, L., 1999, Arcetri Tech. Rep. No 5/99.

Carbillet, M., Femenia, B., Delplancke, F., Esposito, S., Fini, L., Riccardi, A., Viard, E., Hubin, N., Rigaut, F., 1999, in "Adaptive Optics Systems and Tech−nology", R.K. Tyson and R.Q. Fugate eds., (SPIE Proc. 3762, 378−389).

Carbillet, M., Riccardi, A., Fini, L., Viard, E., Delplancke, F., Femenia, B., Esposito, S., Hubin, N., 2000, This conference [P2−32].

Correia, S., Carbillet, M., Barbati, M., Boccacci, P., Bertero, M., Fini, L., Richichi, A., Vallenari, A., 2000, This conference, [P1−12].