# The CAOS Application Builder

Luca Fini, Marcel Carbillet, Armando Riccardi

*Osservatorio Astrofisico di Arcetri, Firenze, Italy*

**Abstract.**

In order to ease the design and implementation of simulation projects a graphical interface has been built on top of the CAOS Simulation Package. The CAOS Application Builder allows a user to build a simulation program (a *project*) by putting together elementary building blocks and specifying the data flow between blocks. When the project has been defined to the user's satisfaction the IDL code which implements the program is automatically generated.

## 1. Introduction

The CAOS software system is a set of software tools specifically designed to allow the modeling of any kind of Adaptive Optics system 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 could concentrate on the scientific aspects of his/her problem, while mundane coding problems were 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 blocks must not be affected too much, and the time efficiency at run-time of the code produced must not be increased significantly.
- The requirements the coding of single blocks imposed by the use of the `AB` must not prevent the use of blocks in the traditional way; i.e.: as usual routines to be called by a user written program.

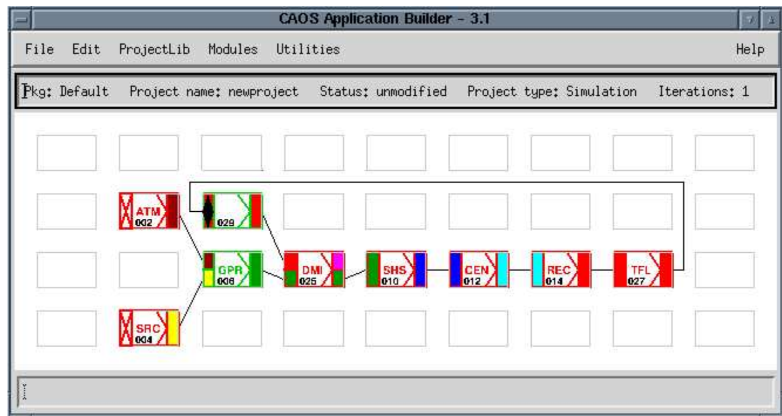


Figure 1. A Project is being built in the Application Builder.

## 2. How the Application Builder Works

When the `AB` is launched it appears to the user, as shown in figure 1, as a graphical window (the **`worksheet`**) provided with a number of rectangular slots and with a number of pull-down menus.

The building blocks, called **modules** (see figure 2) 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 represents the data flow in the program.

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 input (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 particular typed module to display data values of each particular data type. The generic data display module has been designed so that it can check the data type of its input and use the appropriate section of code to display the value of a particular item.

For any module that 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 an 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.

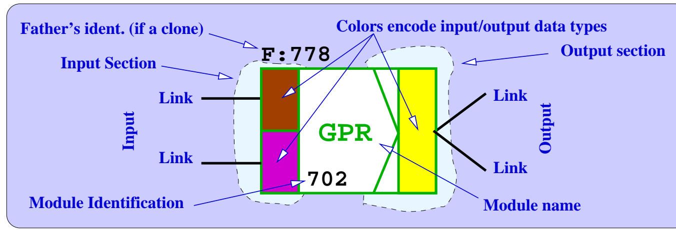


Figure 2. Anatomy of a Module.

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 element.

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

## 3. A few more Goodies

In order to ease the project building, the `AB` includes a few more tools.

The **project library**, is a repository of projects which are distributed together with the CAOS software system which can be merged into a project being developed, or can be used as examples for similar ones.

Moreover the CAOS software distribution can be customized by defining **packages**. A package is a subset of the available modules, possibly specialized for a particular area of applications. By selecting a package the user is set into an environment which is exactly suited to the application to be developed.

## 4. 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 may be later restored in the `AB` for subsequent use.

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

As an example here follows a simplified version of the code generated by the project shown in figure 1:

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

ret=src(O_004_00,src_00004_p,INIT=src_00004_c)
ret=atm(O_002_00,atm_00002_p,INIT=atm_00002_c)

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

```
ret=gpr(0_004_00,0_002_00,0_006_00,gpr_00006_p,INIT=gpr_00006_c)
ret=dmi(0_006_00,0_029_00,0_025_00,0_025_01,dmi_00025_p,INIT=dmi_00025_c,TIME=dmi_00025_t)
ret=shs(0_025_00,0_010_00,shs_00010_p,INIT=shs_00010_c,TIME=shs_00010_t)
ret=cen(0_010_00,0_012_00,cen_00012_p,INIT=cen_00012_c)
ret=rec(0_012_00,0_014_00,rec_00014_p,INIT=rec_00014_c)
ret=tfl(0_014_00,0_027_00,tfl_00027_p,INIT=tfl_00027_c)
```

The `project.pro` procedure skeleton which is “wrapped” around the above code is shown below:

```
RESTORE, 'Projects/luca1/src_00004.sav'   ; Restore parameters
RESTORE, 'Projects/luca1/atm_00002.sav'
RESTORE, 'Projects/luca1/gpr_00006.sav'
RESTORE, 'Projects/luca1/dmi_00025.sav'
RESTORE, 'Projects/luca1/shs_00010.sav'
RESTORE, 'Projects/luca1/cen_00012.sav'
RESTORE, 'Projects/luca1/rec_00014.sav'
RESTORE, 'Projects/luca1/tfl_00027.sav'

@Projects/luca1/mod_calls.pro             ; Initialization

FOR this_iter=1, tot_iter DO BEGIN        ; Main loop
        @Projects/luca1/mod_calls.pro
ENDFOR
```

## 5. Implementation

Because the IDL language was selected as the best choice for the implementation of modules, it was also decided to implement the AB in the same language, although not fully suited to this particular task. It is started as an usual script from the IDL prompt, and it is completely independent from the simulation program it has created: the simulation program can and will run independently on the AB itself.

The final version of the AB is made up of some 39 IDL source files for a total of 6700 lines of code (including full documentation of the source code).

## 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 Technology*, 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, in ASP Conf. Ser., Vol. TBD, Astronomical Data Analysis Software and Systems IX, ed. D. Crabtree, N. Manset, & C. Veillet (San Francisco: ASP), [P2-32],

Correia, S., Carbillet, M., Barbati, M., Boccacci, P., Bertero, M., Fini, L., Richichi, A., Vallenari, A., 2000, in ASP Conf. Ser., Vol. TBD, Astronomical Data Analysis Software and Systems IX, ed. D. Crabtree, N. Manset, & C. Veillet (San Francisco: ASP), [P1-12].