TOWARD A REFERENCE IMPLEMENTATION OF A STANDARDIZED ASTRONOMICAL SOFTWARE ENVIRONMENT

L. Paioro [1], B. Garilli [1], P. Grosbøl [2], D. Tody [3], T. Fenouillet [4], Y. Granet [4], C. Surace [4]

[1] INAF – IASF Milano, via Bassini 15, 20133 Milano, Italy (e-mail **luigi@lambrate.inaf.it**)
[2] European Southern Observatory, Karl-Schwarzschild Str. 2, Garching, 85748, Germany
[3] National Radio Astronomy Observatory / VAO, 1003 Lopezville Rd, Socorro, NM, 87801-0387, USA
[4] Laboratoire d'Astrofisique de Marseille, Université de Provence, Marseille, France

Abstract

The OPTICON Network 3.6 (FP6) and the US NVO, coordinating with international partners and the Virtual Observatory, have already identified high-level requirements and a global architectural design for a future astronomical software environment. In order to continue this project and demonstrate the concepts outlined, the new OPTICON Network 9.2 (FP7) was born and is working on a concrete prototype which will contribute to the development of a reference implementation of the basic core system to be jointly developed by the major partners. As the reference implementation stabilizes, we plan to work with selected groups within the astronomical community to port software to test the new environment and provide feedback for its further evolution. These groups will include both producers of new software as well as the major legacy systems (e.g. AIPS, CASA, IRAF/PyRAF, Starlink and ESO Common Pipeline Library).

1. Current Scenario

For several years the astronomical community has used different data reduction and analysis software which perfectly fitted their scientific needs. Good examples of such software are: IRAF, MIDAS, AIPS, GIPSY and so on, which are still very important tools, widely used today as in the past. However, their old design and lack of modern technologies make it complicated to interact with them and there is scarce support for the new distributed technologies like Virtual Observatory, GRID, HPC, etc. as well as a limited development and maintenance in general.

2. Project Targets

The Future Astronomical Software Environment (FASE) project doesn't aim to be a Virtual Observatory competitor but rather to start from it and create a new software platform for astronomy with extended and even more extensible capabilities. The principal targets are:

a) integration of legacy software within a modern framework that permits to extend their functionalities;
b) easy support and development of new interoperable and distributed applications or simple computational tasks.

3. Networks Working On The Project

The high level requirements and the main architectural concepts have already been defined within the OPTICON Network 3.6 in collaboration with NRAO/USVAO. A white paper describing the framework have been produced as well and distributed with the other document through the project Twiki site [*]. This initial work has been carried on thanks to a specific EU FP6 funding. A new EU FP7 funding has been obtained by OPTICON (leading to Network 9.2) with the purpose of demonstrating the concepts outlined so far with at least one practical implementation meant to create a base for an eventual reference implementation. This implementation comes from a prototyping work with the additional objective of contributing to the architectural design refinement.

4. The Architecture

The architectural design foresees a modular approach (see **Figure A**). This way each single framework component (or set of components) can be developed apart and included in the whole framework as a plugglable module (or package). The development of new framework modules is made easier thanks to the component container paradigm. A container is a sort of wrapper that connects any component (application or computational task) to the framework. The main benefit of a modular architecture is the flexibility and extensibility. The basic framework capabilities can be expanded with new local components or coming from the wide astronomical community.

5. Prototyping FASE

Since the beginning of the FASE project, a parallel prototyping work has been started from a joined effort between INAF-IASF Milano and LAM institutes. Three experimental implementations have been produced so far, the first and the second ones based on DBus, the third based on the emerging SAMP protocol and studied in order to explore the potential of this VO protocol.
With the beginning of OPTICON N9.2, a more formal prototyping work has been started. As starting point, the Milano-Marseille collaboration has significantly contributed to the production of three document drafts concerning the interfaces for the *parameter set* mechanism, the *package manager* and a proposal for a basic implementation. These three documents are now under discussion within the Network and freely accessible via the OPTICON Network 9.2 Twiki site [*]. The present prototyping work has the following short term goals:

a) a basic packaging system and package installer tool;
b) a package manager implementation at least in Python and Java, while a C implementation is planned to be developed afterwards;
c) the definition of the language specific component containers interfaces (Python Java, C/C++);
d) the definition of a recipe containing the minimal set of operations a component developer should perform to put the code in the framework.

A prototype has already demonstrated the FASE architecture concepts with a working example (see section 6) executing a VIMOS Interactive Pipeline Graphical Interface (VIPGI [**]) task. Once the short term goals will be reached, the system will be tested with the ESO Common Pipeline Library, trying to include at least one task in the framework.

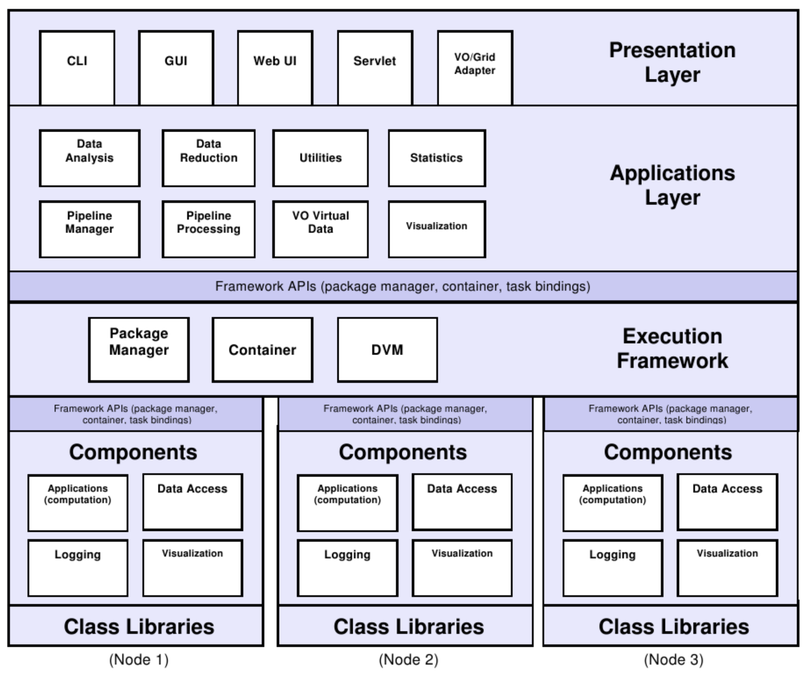

Figure A. The modular architecture of FASE. It can be presented schematically with four layers: the presentation layer (what the user sees), the application layer (what is locally working), the execution framework (the distributed execution engine) and at the final endpoint the components (the actual computational code) which live within their containers.

6. Working Examples

FASE concepts have been already demonstrated with working examples based on the present prototype. Three execution modes have been implemented, for which we report three examples, based on the VIPGI `pipeSpApplyFF` task (managed by a component named `PreRed`), used to perform a preliminary reduction of the VIMOS data:

Example 1: synchronous execution within the Python process (*in-line mode*);
Example 2: asynchronous execution in a distributed context (*distributed mode*);
Example 3: synchronous execution from command-line (*host mode*).

The Milan-Marseille collaboration is working on a set of Java examples as well. If you are interested in the FASE project and/or you wish any other information on the prototype development, please contact the poster author or visit the OPTICON N9.2 Twiki site [*] or the Milan-Marseille prototype site [***].

Example 1:

```
#!/usr/bin/python

from std.system import *
from std.comp import PSet

# SET RUNTIME CONTEXT AS INLINE
pm = RuntimeContext.getPackMan(mode=INLINE)

# LOAD VIPGI PACKAGE
vipgi = pm.loadPackage("pandora.vipgi")

# GET THE LIST OF TASK COMPONENTS
tlist = vipgi.getTaskList()

# THE FIRST TASK IS PreRed, THEN GET IT

# USING THE TASK NAME
prered = vipgi.getTask(tlist[0][0])

# OR USING THE OTYPE
prered = vipgi.getTask(tlist[0][1])

# PRINT THE TASK CAPABILITIES
print prered.getCapabilities()

# DIRECT SYNCHRONOUS INVOCATION OF THE TASK:
# INPUT PARAMETER SET IMPLICITLY DEFINED
# AND PASSED TO THE TASK.
# opset IS THE RESULTING OUTPUT PARAMETER SET
opset = prered(image="st_EG-274_LR_red_Q1_1.fits", \
               msBias="msBias_spec_Q1.fits", \
               msFlat="msFlat_STD_LR_red_Q1.fits")
```

Example 2:

```
#!/usr/bin/python

import time

from std.system import *
from std.comp import PSet

# SET RUNTIME CONTEXT AS DISTRIBUTED
pm = RuntimeContext.getPackMan(mode=DISTRIBUTED)

# LOAD VIPGI PACKAGE
vipgi = pm.loadPackage("pandora.vipgi")

# GET THE TASK DISPATCHER
disp = RuntimeContext.getInvokeDispatcher(mode=DISTRIBUTED)

# SET THE INPUT PARAMETER SET
ipset = PSet(image="st_EG-274_LR_red_Q1_1.fits", \
             msBias="msBias_spec_Q1.fits", \
             msFlat="msFlat_STD_LR_red_Q1.fits")

# ASYNCHRONOUS INVOKE
_id = disp.invokeAsync(prered, "my-tag", ipset)

# WAIT UNTILL TASK STATUS IS "done"
while disp.getIDStatus(_id) != "done":
    print "STATUS ", disp.getIDStatus(_id)
    time.sleep(0.5)

# GET THE TASK EXECUTION RESPONSE:
# THE OUTPUT PARAMETER SET
opset = disp.getReply(_id)
```

Example 3:

```
shell > invoke pandora.vipgi PreRed --help
Usage: pandora.vipgi PreRed (or mos.reduction.prepare) [options]

Options:
  --version         show program's version number and exit
  -h, --help        show this help message and exit
  --ipset=IPSET     Input pset file
  --opset=OPSET     Output pset file
  --image=IMAGE     Input image
  --msBias=MSBIAS   Master bias frame
  --msDark=MSDARK   Master dark frame
  --msFlat=MSFLAT   Master flat frame
shell > invoke pandora.vipgi PreRed --image st_EG-274_LR_red_Q1_1.fits --msBias msBias_spec_Q1.fits
--msFlat msFlat_STD_LR_red_Q1.fits
```

[*] https://www.eso.org/wiki/bin/view/Opticon/WebHome
[**] http://cosmos.iasf-milano.inaf.it/pandora/vipgi.html
[***] http://cosmos.iasf-milano.inaf.it/trac/fase