TASK SYNCHRONIZATION IN THE OBSERVATION CONTROL SOFTWARE FOR THE ESO-VLT CRIRES INSTRUMENT

E. Pozna#, ESO, Garching beim Muenchen, Germany
A. Smette, R. Schmutzer, ESO, Santiago, Chile.

Abstract

The ever increasing pressure for both high spectral and high angular resolution spectrograph imposes an increasing complexity on astronomical instrument control software, now a critical component in the instrument design. To achieve the accuracy required to maintain the image of the target within its 0.2 arcsec entrance slit, the Observation Control software (OS) for the ESO-VLT CRIRES instrument must take into account a number of optical phenomena (differential atmospheric refraction, distortion, etc.), some of them time dependent, even when observing an object moving at a rate different from the object used for auto-guiding. Four internal software control loops adjust the position of mechanical devices and/or the telescope in addition to the OS standard functionalities (e.g. monitoring, exposure handling). Besides internal activities, the OS must promptly response to sequential commands as well as simultaneous interruptions/adjustments from operator via GUI interface. The required advanced synchronization mechanisms are implemented as an extension to the OS framework (a tool collecting the general features of all instrument OS) while allowing for maintainability and future generalization..

INTRODUCTION TO OBSERVATION SOFTWARE AND TO ITS NEW DEMANDS

The Observation Software (OS) of an astronomical instrument is the top level control software that carries out the instructions of astronomers (given as sequential command series) in order to record astronomical images. Such instrument is the VLT CRyogenic high-resolution InfraRed Echelle Spectrograph (CRIRES).

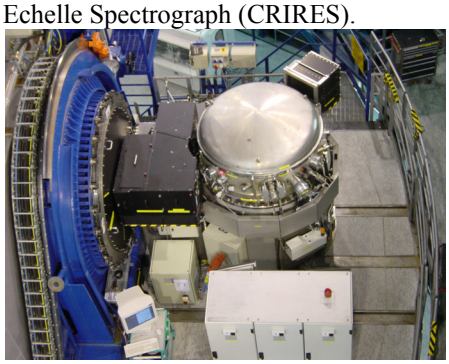

Figure 1: Instrument CRIRES [1].

Receiving a command the OS distributes the necessary actions to the subsystems (detectors and groups of mechanical devices) and the telescope and synchronises their actions. The main responsibility of OS is to take care of series of exposures and meanwhile monitor the system giving up-to-date information to the operator. These common requirements of OS are well supported by the BOSS framework [2]. BOSS gives such level of support that it fulfils all requirements of a simple instrument. The control software of CRIRES is however far from the simple cases; its many extra (and complex) functionalities rose problem with synchronisation and event queues at the top level control software.

REQUIREMENTS AND ANALYSIS OF CRIRES OS

CRIRES OS also has additional elements to control (see Fig. 2.) than an average instrument OS, nevertheless the difficulties arise not from this but from its manifold

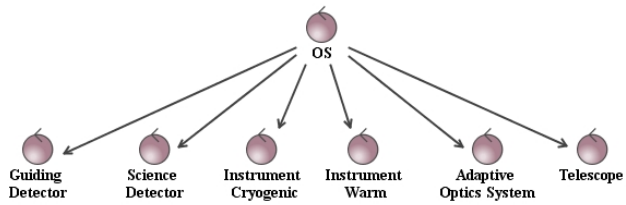

Figure 2: Subsystems controlled by CRIRES OS.

functionalities (listed on Fig. 3.). Setting aside the pure calculation procedures (chromatic effects, distortion, refraction) we can group the functionalities as follows:

- User-triggered setting of hardware elements: used during initial setting or to intervene in the operation. E.g. offsetting the telescope, setting the derotator angle, setting filter.
- Internal periodic actions: carrying out fine positioning of mechanical devices (differential tracking; guiding, i.e. sending offsets to telescope or adaptive optics field selector based on offsets measured by the slit viewer detector control system; adjusting the environmental changes –e.g. airmass- via periodic refraction calculation; adaptive optics offload)

The various fine tuning loops all have different frequencies, and can be switched on/off by the operator.

Interdependencies between Functionalities

The user actions and internal loops involve the movement of instrument and/or telescope devices and often the execution of other associated activities. The scope of this paper does not allow for discussion of all functionalities in detail; the examples below are to illustrate their connectivity:

- Changing the slit viewer filter (i.e. the effective wavelength) imposes a change in the differential refraction between the effective wavelength of the

#epozna@eso.org

slit viewer (used for secondary guiding) and the observing wavelength, which requires an offset to be sent to the telescope or the adaptive optics field selector. Filter change also imposes an update on the real time display which, besides showing the real stars also shows their expected locations, together with other elements, such as guiding window position.

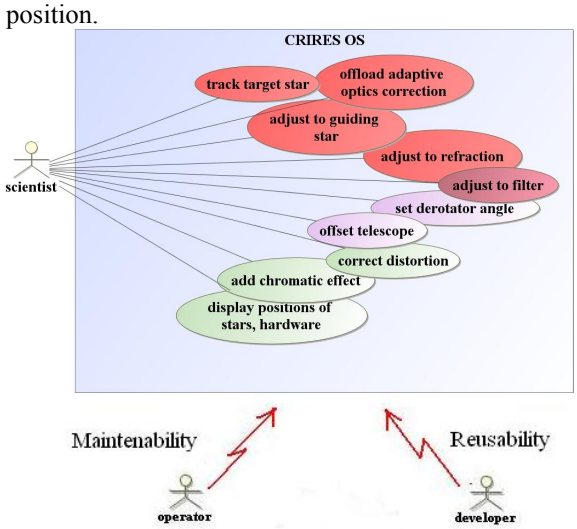

Figure 3: Main functionalities of CRIRES OS.

- During refraction compensation the Guiding loop should be suspended and vice versa.
- Refraction compensation is also part of the differential tracking, during which all other internal loops should be suspended.
- When differential tracking is due it should get priority amongst the other internal loops.

Asynchronous Message Handling

Positioning of motor devices especially telescope can be time consuming, therefore asynchronous message handling is recommended during the positioning actions not to hang the OS from its other activities (e.g. monitoring) and remain responsive. The aim is to handle events during a slow action, however in case of CRIRES there are events that must not be dealt with simultaneously. The execution of a colliding event must be delayed (e.g. user sending a command during the process of an internal loop).

Event Queue Dilemmas

Using sequential event handling, event queues are unavoidable. A growing event queue e.g. due to a possible slow (synchronous) action or simultaneous events raises further matters to consider in case of CRIRES:

- What if an event on the queue loses its validity by the time it gets executed? This can happen for example if the guiding correction –calculated periodically by the detector- gets delayed due to another previously delayed positioning action. This kind of behaviour would make the target jump around its desired location, making the system unreliable.
- How to ensure that the event queue is limited and priority tasks are executed first (ensuring a responsive system).
- A multiple occurrence of an event (that is generated periodically) on the event queue would result in an unnecessary frequency of its execution once the resources are freed.

Parallel Commands during Operation

While executing predefined scripts of sequential commands – which is the typical way of carrying out observations - , the operator or the astronomer might wish to interact with the system via GUI; e.g. switching ON/OFF internal actions, or move elements. This means to permit the handling of simultaneous commands, which by default is rejected by BOSS.

Robustness, Maintainability, Reusability

Object oriented techniques offer robust, maintainable and reusable solution that is required by both the operator and the developer. However the key to achieving this is the lack of interdependency!

IMPLEMENTATION

In order to resolve the seemingly contradictory problems the following aspects were put into view:

- Remove interdependency by breaking down the individual functionalities into a list of reusable components (referred as actions).
- When an event is caught, add its sequence of actions (i.e. the id-s of the actions) to the ‘action-list’.
- Execution of the list of actions should be managed by a supervisor.

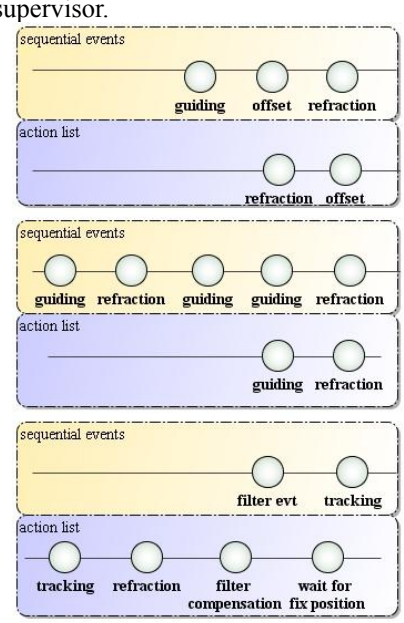

Figure 4: Three examples of event queue handling. Top row shows the events collected during a slow action (when left event is the most recent); bottom row shows the actions to be executed taking into account all events.

- Map the actions with Id-s so that a non-supervisor can refer to them.
- Change the course of actions already on the list according to the latest event.
- Set a maximum occurrence for each action to protect event queue from growing indefinitely.

Software Design

The implementation of the individual actions is based on a common abstract class 'ACTION' (Fig. 5.) to ensure the capability of their general handling. The event callback function is placed in its dominant action class (though it could be now separated). This callback function

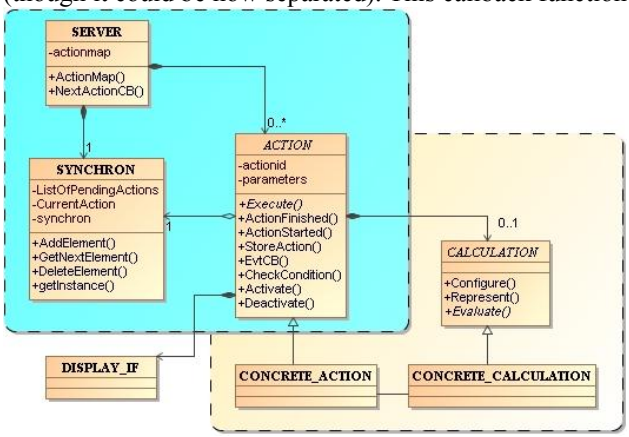

Figure 5: Software design.

includes the check on the currently running and pending actions via the associated SYNCHRON class. During the event handling the pending action list is updated (existing list might be modified, new actions might be added, see examples on Fig. 4.). Should the system be idle and the action list empty at the moment when the event is received, then the execution of the first action is initiated.

The heart of this design is the singleton class SYNCHRON which stores the information (id and parameters) about the pending actions, yet independent from the ACTION classes. The SYNCHRON class implements an iterator on the list of pending actions. This list is allowed to be dynamically modified via its functions (e.g. delete all action of given type, relocate action, update action with new parameter). (During the execution of a command the currently running action is set accordingly in order to prohibit other commands being executed. This is relevant for non-blocking asynchronious commands.)

The maximum number of occurrences of an action on the list can be easily established. For example SETUP and OFFSET commands may occur twice, where the limit comes naturally from the operational conditions; one request from GUI and one by the observational script. Any additional commands will be rejected.

The ACTION class is typically part of a group of cohesive classes, i.e. it is associated with a calculation object. The calculation class is responsible for the core evaluation of optical phenomena e.g. refraction, data handling (e.g. in case of filter compensation, ephemerides, chromatic effect).

The calculation classes are stable, in terms of being instrument independent while the actual CONCRETE_ACTION class (which is typically responsible for updating positions of hardware devices) contains the instrument specific information.

When an action is terminated it generates a new event to signal the superior SERVER class to execute the next action if there is any. This internal event based solution makes it possible to catch outside events even if the pending action list contains blocking actions only. The system is now capable to handle all type of actions (commands, events, non-message, complex messages, synchronous asynchronous, periodic etc.) in the same way. It also allows the handling of monitoring or emergency procedures outside the synchronization mechanism enabling to be used as an extension to the framework BOSS.

LESSON LEARNT AND FUTURE CONSIDERATION...

One of the most challenging parts of the project was to identify the possible source of problems, that even if unhandled may remain hidden during tests, but can cause disturbance during operations.

The system described above has been in operation (see Fig. 6.) for several months without any break down, and offers an easy way for future updates (e.g. adding additional loops). The software design created can be also easily turned into a reusable framework.

The authors of this paper believe that CRIRES software might be just the first of its kind at ESO. The increasing resolution of the detectors imposes higher demand on the control aiming to achieve (and/or not to loose) the level of precision that the new detectors are now allowing.

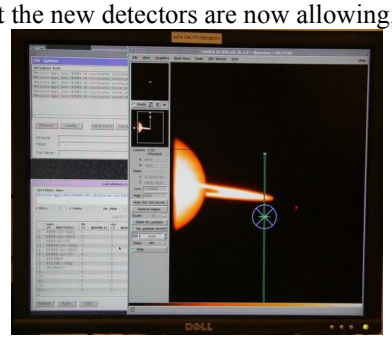

Figure 6: First test result with Saturn showing the action list in the background during the offset of the telescope.

REFERENCES

[1] H. U.Käufl at all, "CRIRES: commissioning and first science results" Proceedings of SPIE Vol. 7014 (SPIE, Bellingham, WA 2008) 70140W.

[2] E. Pozna, G.Zins, P.Santin, S.Beard, "A common framework for the observation software of astronomical instruments at ESO", Proceedings of SPIE Vol. 7019, 70190Q (2008).