SHARK-NIR: implementation of the instrument control software SHINS

De Pascale, Marco, Baruffolo, Andrea, Salasnich, Bernardo, Ricci, Davide, Briegel, Florian, et al.


Event: SPIE Astronomical Telescopes + Instrumentation, 2020, Online Only
SHARK-NIR: Implementation of the instrument control software SHINS

Marco De Pascalea, Andrea Baruffoloa, Bernardo Salasnicha, Davide Ricci, Florian Briegelb, Jacopo Farinatoa, Federico Biondi, Paul Grenzd, and Daniele Vassalloa

aINAF - Padova Astronomical Observatory, Vicolo dell’Osservatorio 5, I-35122 Padova, Italy
bMax Planck Institut, Koenigstuhl 17, D-69117 Heidelberg, Germany
cMax Planck Institut, Karl-Schwarzschild-Str. 1, D-85741 Garching, Germany
dUniversity of Arizona, 993 N. Cherry Ave, Tucson, AZ, 85721, USA

ABSTRACT

The System for coronagraphy with High Order adaptive optics in Z and H band (SHARK-NIR), is a high contrast imager with coronagraphic and spectroscopic capabilities, which will be mounted at the Large Binocular Telescope (LBT). We describe the implementation of SHINS, the SHARK-NIR control software, mainly realized with TwiceAsNice framework from MPIA - Heidelberg and ZeroC-ICE framework. We describe how we implemented the software components controlling instrument subsystem as well as the adaptation of already tested libraries from other instruments at LBT, such as LINC-NIRVANA.

Keywords: Control Software, Implementation, Coronagraphy, Adaptive Optics, Intrumentation, Telescope

1. INTRODUCTION

The System for coronagraphy with High Order adaptive optics in Z and H band (SHARK-NIR), is a high contrast imager with coronagraphic and spectroscopic capabilities, which will be mounted at the Large Binocular Telescope (LBT). It will observe in the near infrared, between 0.96 and 1.7 microns. Its main scientific goal is the direct imaging of exo-planets, their detection and characterization, taking advantage of the adaptive optics offered by LBT. Other science objectives include brown dwarfs, protoplanetary discs, stellar jets, QSOs and AGNs.

In this paper we describe the implementation of the SHARK-NIR instrument control software, named SHINS. As described in the design paper, SHINS is implemented using two frameworks: TwiceAsNice (TAN) provided by MPIA - Heidelberg, and ICE (Internet communication engine) provided by ZeroC company. More in detail, we describe the implementation of software components responsible for controlling the instrument subsystems, as well as how we adapted libraries already used and tested on other instruments at LBT, namely LINC-NIRVANA and ARGOS.

2. THE INSTRUMENT

Fig. 1 shows the synoptic diagram of SHARK-NIR; light arrives from the LBT Interferometer focus, thanks to a deployable motorized mirror at the instrument entrance, and is then imaged on the H2RG by Teledyne IR detector hosted in the cryostat on the opposite side. Along the optical path are an internal deformable mirror, used to compensate for high frequency vibrations (order of 1 KHz), the atmospheric dispersion corrector, a neutral density wheel and technical camera acting as WFS for the DM. Of great importance for this instrument are three aperture wheels used for coronagraphy, with two of them used also for spectroscopy, then two wheels mounting the scientific filters, a deployable lens to image the instrument pupil on the scientific camera and the wheel mounting dual band filters; the instrument is mounted on its own derotator. The majority of device building the calibration unit are on the instrument optical bench: a deployable mirror, an integrating sphere and an additional

Further author information: (Send correspondence to M. D. P.)
M. D. P.: E-mail: marco.depascale@inaf.it, Telephone: +39 049 829 3494
A. B.: E-mail: andrea.baruffolo@inaf.it
linear axis which inserts a fiber in the calibration unit optical path; the three lamps feeding the calibration unit are all in the instrument rack. SHARK-NIR is built by a consortium of institutes: the Astronomical Observatory of Padova for INAF, the Max-Planck Institute of Hidelberg and the Steward Observatory for the University of Arizona.

3. SHINS DESIGN AND ARCHITECTURE

We leave the description of SHARK-NIR control network architecture to a previous paper,\textsuperscript{1} in this section we recall from that same paper the design and architecture for SHINS, adding the few modifications occurred in the meantime.

3.1 Design

The design of SHINS is similar to that of ESO VLT control software: a central component is in charge of dispatching commands to peripheral components, dedicated to subsystems control, while all of the observation, calibration and maintenance procedures are implemented by means of template scripts. The execution environment is divided in two, as mention in Sec. 1 and as shown in Fig. 2, and is developed with two different frameworks: TwiceAsNice and ZeroC ICE. There were two reasons for this choice: the MoCon control electronics\textsuperscript{6} custom made by MPIA - Heidelberg, and the control software for the scientific camera, developed using the INDI (Instrument-Neutral Distributed Interface) protocol;\textsuperscript{7} we deemed ZeroC ICE, on top of which TAN is developed, a good encounter point between TAN and INDI.
3.2 Architecture

With reference to Fig. 2, following the design we broke down SHINS in subpackages, or subcomponents, each responsible to control a subsystem. We devised the following sub-packages:

- **seq**: the sequencer component, interfaced to the LBTO observation preparation tool. It communicates the sequence of operations to be executed to **obs_ctrl**;
- **tiptilt_ctrl**: interfaced to real-time tip/tilt control loop software;
- **tcs_ctrl**: interfaced to the telescope control system, will ask for preset and offset, for trajectory information to drive the derotator, and AO commands;
- **track_ctrl**: controls the two tracking functions of derotation and atmospheric dispersion corrector (ADC);
- **motion_ctrl**: controls all the non-tracking motorized functions, such as the wheels and the deployers;
- **calunit_ctrl**: controls the calibration unit devices;
- **data_mgr**: merges FITS headers from SHINS and from the scientific camera, passing the final FITS file to the LBTO Archive for ingestion.
With reference to the previous paper describing design and architecture in detail, the interface using the INDI protocol to communicate the subpackages and devices status to LBTO has been removed, these information will be directly written to LBTO dictionary, through Telescope Control System (TCS) interface.

4. SHINS IMPLEMENTATION

The control part of SHINS is implemented mainly in C++, few parts of SHINS tasks are executed by small Python scripts called from inside the main code, mainly for ease of implementation; all of the templates scripts, used to execute observation, calibration and maintenance procedures are implemented in Python.

Starting with the Ice execution environment, the central component, data manager and sequencer are implemented as three separate processes; to each subsystem in the TAN execution environment corresponds a TAN service, giving access to the underlying devices.

4.1 The central component

As per design, the central component obs_ctrl acts as a server application, receiving the sequence of operations from the sequencer and dispatching the commands to the subcomponents. In ZeroC Ice framework, this is obtained by declaring obs_ctrl as an Ice object, using the declarative Specification Language for Ice (SLICE); to such object, interfaces with operations can be added. For obs_ctrl we declared a main interface to which the sequencer subpackage will connect, and that implements all the operation necessary for SHARK-NIR to operate. Listing 1 reports part of the SLICE declaration for the central component; to reduce clutter, we defined a main module sharknir, defining a namespace, in which to declare common data structures that will be used in operation signatures. In sharknir module, a second module obsCtrl is defined, with its main interface Sequencer, of which Listing 1 reports some of the declared operations: setup motion devices, enable the real-time tip/tilt subsystem and upload to its deformable mirror a specific shape (so to reduce NCPA), setup and expose the scientific camera, send the final FITS file to data_mgr for archiving and finally preset the telescope.

As mentioned above, SLICE language is purely declarative, meaning that it expresses the logic of a computation, what it should do: it is not possible to write executable statements. For this reason SLICE declaration must be compiled to generate executable source code; we compiled our declaration to C++, since the C++ implementation of TAN libraries. Compilation of SLICE declaration of obsControl generates the skeleton class Sequencer, a purely virtual C++ class in our case; such class in Ice is called a servant, and for each interface a servant is generated. Servants make available to clients all of the interfaces operations. The real implementation of the Sequencer interface operations is done in a class, SequencerI, derived from the virtual one introduced before; in Listing 3 is reported an extract of the implementation for C++ declaration of SequencerI.

In Listing 3, in addition to C++ method signature of Ice operations reported in Listing 1, the private section of SequencerI class shows four member objects: m_motion, m_calibration, m_track and m_tiptilt; through these objects SequencerI class gains control of the related subpackages, these classes are described more in detail in Sec. 4.2; Fig. 3 reports the class diagram used to implement SequencerI class.

4.2 TAN Related Subpackages

With reference to Fig. 2, TAN related subpackages are those used to control all of the motion devices, to interface with the real-time subsystem and to the telescope control software. In the following sections we will describe the approach we used to implement the control of such devices. A common feature to all the subpackages developed in the TAN framework is that each subpackage in mapped to a TAN service available through the TAN server process, each service controlling a subset of SHARK-NIR devices; each of these services is defined by a configuration file for TAN, reporting, among information for the low level connection (e.g. host and port), the list of available devices (see Listing 4 for an example).
```cpp
#include <LbtoIIFInterface.ice>

module sharknir {
    dictionary<string, string> dictSetup;

    module obsCtrl {
        // ...
        interface Sequencer {
            // INSTRUMENT WIDE OPERATIONS
            int setupInstrument(sharknir::dictSetup dictSetupNamedPos);
            // ...
            // REAL-TIME TIPTILT OPERATIONS
            int RTCTTEnableLoop(sharknir::dictSetup dictTipTiltSetup);
            int RTCTTModeUpload(lbto::SeqModes modes);
            // ...
            // SCIENTIFIC CAMERA CTRL OPERATIONS
            int sashaSetup(sharknir::dictSetup dictSashaSetup);
            int sashaExpose(sharknir::dictSetup dictSashaSetup);
            // ...
            // DATA MANAGER OPERATIONS
            int datamgrArchiveFITS(string fitsFileAbsName);
            // ...
            // TELESCOPE CONTROL SYSTEM OPERATIONS
            int PresetTelescope(sharknir::dictSetup dictTelSetup);
            // ...
        }
    }
}
```

Listing 1: Skeleton of SLICE definition of `obs_ctrl`, showing the two implemented interfaces and one operation among those defined. The capitalized comments are to point the different sections in the interface, one per subpackage. The inclusion of `LbtoIIFInterface.ice`, gives access to data structures (such as `lbto::SeqModes`) and operation implemented for the TCS interface.

```cpp
#include <LbtoIIFInterface.ice>

module sharknir {
    dictionary<string, string> dictSetup;

    module obsCtrl {
        // ...
        interface Sequencer {
            // ...
            // ...
        }
    }
}
```

Listing 2: Endpoint definition and use Two listing are show: on top is an extract of the Ice properties file used by `obs_ctrl` reporting the endpoint definition used by clients to connect. The bottom list is an extract of C++ code from `obs_ctrl` defining the adapter accessible from clients from the endpoint specified above; following is the addition to the adapter of the `SequencerI` servant.

```cpp
#include <LbtoIIFInterface.ice>

module sharknir {
    dictionary<string, string> dictSetup;

    module obsCtrl {
        // ...
        interface Sequencer {
            // ...
            // ...
        }
    }
}
```

Listing 2: Endpoint definition and use Two listing are show: on top is an extract of the Ice properties file used by `obs_ctrl` reporting the endpoint definition used by clients to connect. The bottom list is an extract of C++ code from `obs_ctrl` defining the adapter accessible from clients from the endpoint specified above; following is the addition to the adapter of the `SequencerI` servant.

```cpp
#include <LbtoIIFInterface.ice>

module sharknir {
    dictionary<string, string> dictSetup;

    module obsCtrl {
        // ...
        interface Sequencer {
            // ...
            // ...
        }
    }
}
```

Listing 2: Endpoint definition and use Two listing are show: on top is an extract of the Ice properties file used by `obs_ctrl` reporting the endpoint definition used by clients to connect. The bottom list is an extract of C++ code from `obs_ctrl` defining the adapter accessible from clients from the endpoint specified above; following is the addition to the adapter of the `SequencerI` servant.
using namespace sharknir;

class SequencerI : public obsCtrl::Sequencer {

private:
    motionCtrl m_motion;
    calunitCtrl m_calibration;
    trackingCtrl m_track;
    tiptiltCtrl m_tiptilt;

public:
    // Constructor
    // Destructor
    virtual Ice::Int setupInstrument (const dictSetup& _dictSetupNamedPos,
                                           const Ice::Current&);

    virtual Ice::Int RTCTTEnableLoop (const dictSetup& _dictTipTiltSetup,
                                       const Ice::Current&);

    virtual Ice::Int RTCTTModeUpload (const lbto::SeqModes& _modes,
                                       const Ice::Current&);

    virtual Ice::Int sashaSetup (const dictSetup& _dSashaSetup,
                                 const Ice::Current&);

    virtual Ice::Int sashaExpose (const dictSetup& _dSashaSetup,
                                const Ice::Current&);

    virtual Ice::Int datamgrArchiveFITS(const std::string& _absFileName,
                                         const Ice::Current&);

    virtual Ice::Int PresetTelescope (const dictSetup& _tel_sup,
                                       const Ice::Current&);

};

Listing 3: Declaration of C++ class implementing sequencer interface of obs_ctrl Ice object. The C++ signature for one operation is also shown (see Listing 1 for its definition in SLICE language). Also the four member objects to control the related subsystems are reported. All of the method have a const Ice::Current parameter, this is mandatory and holds all the information on the currently executing request and is passed to the skeleton method.

ADAPTER:Endpoint = [NAME=motion_ctrl-svr, PORT=50200, TIMEOUT=10]
# ... MOTION_CTRL.WHEELS.APODIZER_W.CONFIG = motion_ctrl/wheels/motion_ctrl.wheels.apodizer_w-svc.cfg
MOTION_CTRL.WHEELS.ND_FILT_W.CONFIG = #<path/to/nd_filt_wheel/config>
# ...
MOTION_CTRL.BEAM.PUPIL_LENS_DEP.CONFIG = motion_ctrl/beam/motion_ctrl.beam.pupil_lens_dep-svc.cfg

Listing 4: Extract of TAN configuration file for motion_ctrl service; this service runs all devices related to non-tracking motion functions. The firs line gives information on the endpoint access, such as the port and the endpoint name; following are three TAN properties related to the available devices: two wheels and a linear stage. Files to which the defined properties points give information on the hardware devices.
Figure 3. Class diagram of **SequencerI** class, reporting its parent class and classes of the most important members, related to subsystems control, with indication of the two frameworks used. Objects to control motion devices under TAN framework, **multiStageX**, **multiStageXY** and **multiWheel**, are all implementation of the same virtual class, since they perform the same operations, but they are interfaced to different kind of device, thus using types of TAN objects.

### 4.2.1 Motion devices components

Fig. 3 is shows the class diagram for **SequencerI** class, which implements all of the methods to control SAHRK-NIR. The control on devices attached to MPIA motion control electronics is obtained in **SequencerI** through three member objects: **motionCtrl**, **calunitCtrl** and **trackingCtrl**; each of these objects connects to the TAN services running the pertaining devices.

Lets consider **motionCtrl** C++ class, which models **motion_ctrl** subpackage; to control the underling devices we developed three classes, **multiStageX**, **multiStageXY** and **multiWheel**, one for each type of device (the 2-axis device is the entrance motorized mirror in Fig. 1) all implementing the methods of the purely abstract class **multiDevice**. The choice of a purely abstract parent class was driven by the fact that all devices has to
perform the same operations, while they are mapped in TAN to different C++ objects; using a template class with template type the TAN object turned out to add too much complexity.

### 4.2.2 Real-time tip/tilt subsystem component

The real-time tip/tilt subsystem is composed by a 97-actuators deformable mirror on board of SHARK-NIR, a WFS near infrared camera and a basic control unit (BCU) reading imaged from the WFS, running calculation and applying the tip/tilt shape to the DM in real-time, so to reduce telescope vibrations; also, to reduce NCPA, it is possible to upload a static shape to the DM, kept during the tip/tilt loop. All of the communication happens through the BCU, there is no direct communication with DM electronics or WFS.

The private company that produced the BCU, Microgate, provided us with a control software made by a set of MATLAB®, The logic of such software is to write directly in the BCU memory the required quantities, sending them from the workstation to the BCU with UDP packets using the Microgate Protocol.

We implemented the real-time tip/tilt control component in two stages: we used the MPIA implemented C++ TAN libraries for ARGOS to communicate at low level with the BCU, that is preparing the UDP packets and send them to BCU. The higher level part, BCU DM and WFS initialization, has been implemented as a porting of the MATLAB code to C++.

### 4.2.3 Telescope control software component

The Large Binocular Telescope provides a set of API for instruments control software to interface to TCS, the Instrument InterFace (IIF), such APIs are declared in SLICE and implemented in C++. The IIF APIs are accessible to SHINS thanks to a TAN service (`tcs_ctrl` in Fig. 2) that we implemented using C++ libraries developed and tested by MPIA for LINC-NIRVANA: the service is run by the TAN server in a separate process with respect to SHINS central component; `obs_ctrl` connects directly to it by means of a Nice Application object, much similar to ZeroC Ice Application.

### 4.3 Data Manager

Data manager (`data_mgr` in Fig. 2) is responsible for merging FITS headers and send the final FITS with data and full header to LBTO archive for ingestion.

For what concerns header merging, the FITS file for the observation produced by the scientific camera is received by `obs_ctrl` through the INDI interface (see Fig. 2). This is implemented by listening to INDI TCP port, receiving the file as raw bytes and writing them to disk; in this process is not possible to add new FITS keywords to the header, thus a second file is created by `obs_ctrl`, with no data and with header populated by all the required FITS keywords. The `data_mgr` component will receive this last file and the FITS file produced by the scientific camera. Moreover this component will merge the scientific camera header in the full header produced by `obs_ctrl` as well as copying the data, using CFITSIO APIs.

### 4.4 Sequencer and Templates

Sequencer (`seq` in Fig. 2) is the only component fully implemented in Python, and it is responsible for running templates scripts which contains all of the operations to be performed by the instrument; template scripts are implemented as well in Python as scripts accepting parameters. Values for parameters of template scripts are specified by the observer and provided to the specific script by means of a XML formatted file; the `seq` component is responsible for parsing the XML, extracting both parameter values and template script to execute.

Since the Ice declaration of `obs_ctrl` is compiled also in Python, an interface in this language to all operations of the central component is available; to access such operations each template make use of the Ice adapter defined in Listing 2. The code implementing client access to `SequencerI` interface methods is reported in Listing 5: line 1 defines the adapter endpoint as in Listing 2, while the following instructions initialize the Ice Communicator object `ic` to communicate with the Ice session and access the adapter.
Listing 5: Extract of Python code for client connection to `Sequencer` interface operations of `obs_ctrl` component, by means of an Ice proxy giving access to adapter defined in Listing 2.

5. FINAL REMARKS

SHARK-NIR is a coronagraphic instrument that will be mounted at LBT with the main scientific goal of finding and characterizing exoplanets. In this paper we presented some aspects of the implementation of its control software, SHINS; following the final architecture in Fig. 2, we described the implementation of the central component, made in C++ inside ZeroC Ice, thus using its SLICE language. We also described how are implemented the software components controlling motion devices, both tracking and non tracking, done using C++ TwiceAsNice framework from MPIA; we as well explained the implementation of the real-time tip/tilt component, the interface to TCS, how operates the data manager component and finally the component developed in Python, the sequencer, with a view on templates script.

SHARK-NIR is now in its final integration stage and will be subject at LBT review for acceptance in March 2021.

REFERENCES


