

Design and implementation of a service-oriented driver architecture for LINC-NIRVANA

Frank Kittmann¹, Florian Briegel¹, Lars Mohr¹, Sebastian Egner²,
Wolfgang Gaessler¹, Juergen Berwein¹, Alexey Pavlov¹, Clemens Storz¹

¹Max-Planck-Institute, Koenigstuhl 17, 69117 Heidelberg, Germany

²National Astronomical Observatory of Japan, Subaru Telescope,
650 North A'ohoku Place, Hilo, HI 96720, USA

ABSTRACT

LINC-NIRVANA (LN) is a German-Italian Fizeau (imaging) interferometer for the Large Binocular Telescope (LBT). The Instrument Control Software (ICS) of this instrument is a hierarchical, distributed software package, which runs on several computers. In this paper we present the bottom layer of the hierarchy - the Basic Device Application (BASDA) layer. This layer simplifies the development of the ICS through a general driver architecture, which supports different types of hardware. This generic device architecture provides a high level interface to encapsulate the hardware dependent driver. The benefit of such a device architecture is to keep the basic device-driver layer flexible and independent from the hardware, and to keep the hardware transparent to the ICS. Additionally, the basic device-driver layer supports interfaces to IDL based applications for calibration and laboratory testing of astronomical instruments, and interfaces to engineering GUIs that allow to maintain the software components easily.

Keywords: LBT, LINC, NIRVANA, MPIA, ICS, Instrument Control Software, Basic Device Application

1. INTRODUCTION

LINC-NIRVANA¹ (Large Binocular Telescope Interferometric Camera - Near-IR / Visible Adaptive Interferometer for Astronomy) is a near infrared image beam combiner with advanced, multi-conjugated adaptive optics (MCAO) for the LBT² on Mount Graham in Arizona. The instrument will combine the light coming from the two 8.4m primary mirrors of the telescope in "Fizeau" mode to achieve a resolution equal to that of a 23m single mirror telescope.

A field from 2 to 6 arcminutes diameter of the two beams, which just have entered the LINC-NIRVANA instrument, are redirected by an annular mirror into the Ground-layer Wavefront Sensors (GWS). These devices measure the turbulence directly above the telescope sensing up 12 natural guide stars using 12 Star Enlargers (SE), 2 motor stages each (for a total of 24 motorised linear stages). The CCD captures the superimposed Star Enlarger pupils and send the correction signals to the adaptive secondary deformable mirrors with 672 actuators^{3,4} each to compensate the distortions due to the atmospheric turbulence. The detector is mounted on a remotely controllable linear stage for focusing. The whole unit (24 Star Enlargers and the detector) is attached to a rotation unit to follow the sky.

The inner part of the light, the central two arcminutes of the field of view, is collimated to produce an envelope of constant diameter. The collimator device consists of two Collimator Focus Stages and two Xinetics Deformable Mirrors⁵ (DM). The Xinetics DMs, with 349 actuators each, are optical conjugated to the high-layer turbulence (8-15km) and also the mid-layer turbulence (4-8km). Both are moveable mounted along the optical axis to focus on the optimal altitude of turbulence.

Further author information: (Send correspondence to Frank Kittmann)

Frank Kittmann: E-mail: kittmann@mpia.de, Telephone: +49 (0)6221 528 409

Wolfgang Gaessler: E-mail: gaessler@mpia.de, Telephone: +49 (0)6221 528 277

Florian Briegel: E-mail: briegel@mpia.de, Telephone: +49 (0)6221 528 256

A dichroic mirror directly behind the Collimator reflects the visible light toward the Mid-High Layer Wavefront Sensors (MHWS). Inside of the MHWS, the beam is divided by a moveable Fold Mirror to detect the guide stars via the patrol camera and to place the 8 Star Enlargers directly to the position of the guide stars. On the other hand a second detector takes the pupil images of the Star Enlargers and send the corrected signals to the Xinetics DMs in order to correct the wavefront of the mid-high layer. A rotating K-Mirror in front of the MHWS is used to derotate the sky movement.

The near-infrared radiation pass the dichroic and enters the cryostat, where the science sensor - a Cassegrain camera - takes the final interferometric image. A fringe tracking sensor detects the differential atmospheric piston and controls a mirror, which is attached to a piezo-electric actuator. The different optical path length can be corrected by shifting the mirror sideways to increase one optical path while reducing the other one.

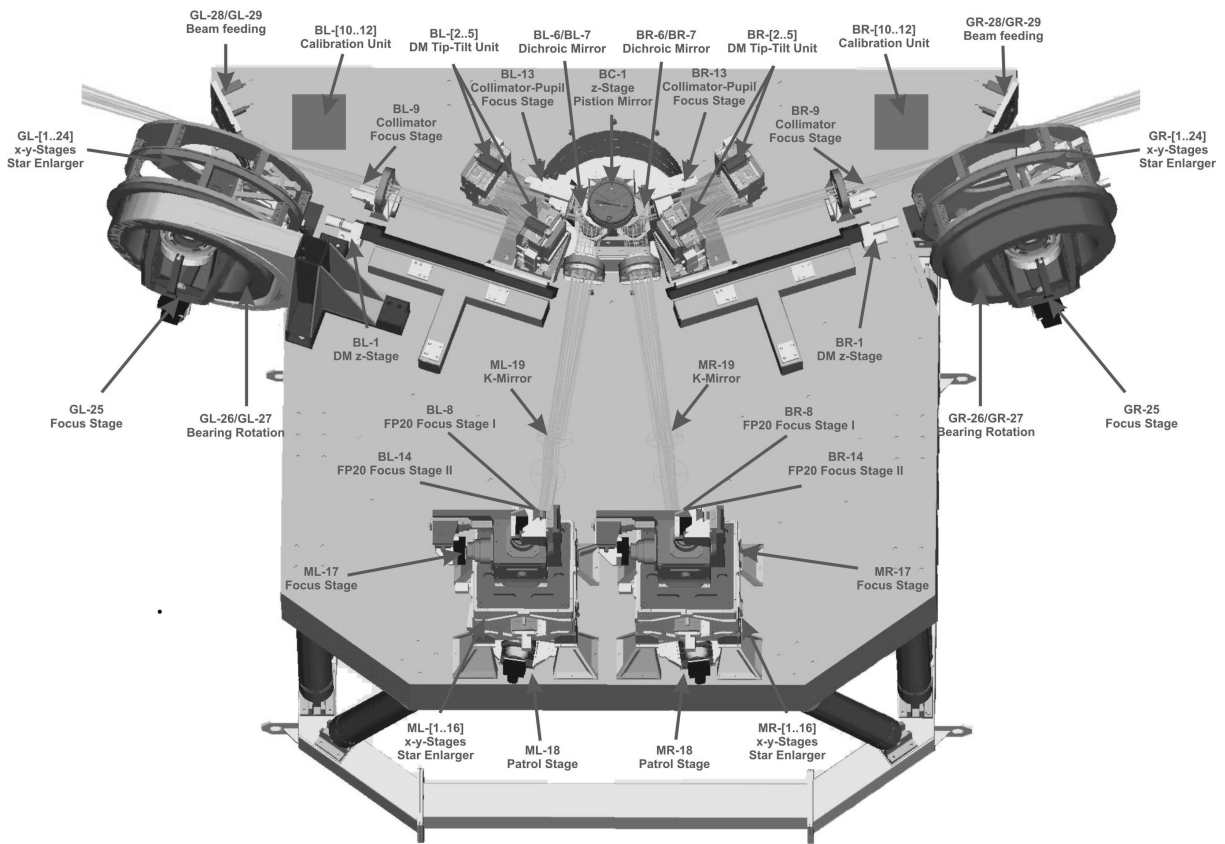


Figure 1. LINC-NIRVANA bench with motor labels

The complexity of the opto-mechanical design (Figure 1) will be reflected also in the Instrument Control Software.⁶ The entire Instrument Control Software of LINC-NIRVANA is structured in three layers. The top layer is the System Layer containing the master ICS. It manages the ICS sub-systems in the middle layer (Sub-System Layer) including the sub-system dependent services. The bottom layer covers the device driver and has a direct connection to the hardware. The wide range of different hardware devices, used in the LINC-NIRVANA instrument, force the development of a coherent device wrapper for the hardware dependent software interfaces. The focus of this work is to present the concept and the design of the device wrapper.

2. THE BASIC DEVICE APPLICATION PACKAGE

The Basic Device Application Package (BASDA) refers to the coherent device wrapper and is a container that collects the hardware dependent drivers. It encloses the low level hardware interfaces and provides hardware independent high level interfaces to the upper layer. The hardware independent high level interfaces are based on generic device interfaces, which are defined once and are available for the sub-system dependent services. This kind of software implementation keeps the software flexible for possible hardware changes in the future and extendable to support new types of hardware easily.

2.1. BASDA Concept

The LINC-NIRVANA software architecture is based on a distributed software design. The software runs on several computers and allows a decentral control of the LINC-NIRVANA hardware devices. The challenge of the BASDA package is to develop a software that allows a remote control of the hardware devices by providing the hardware device interfaces into the local area network.

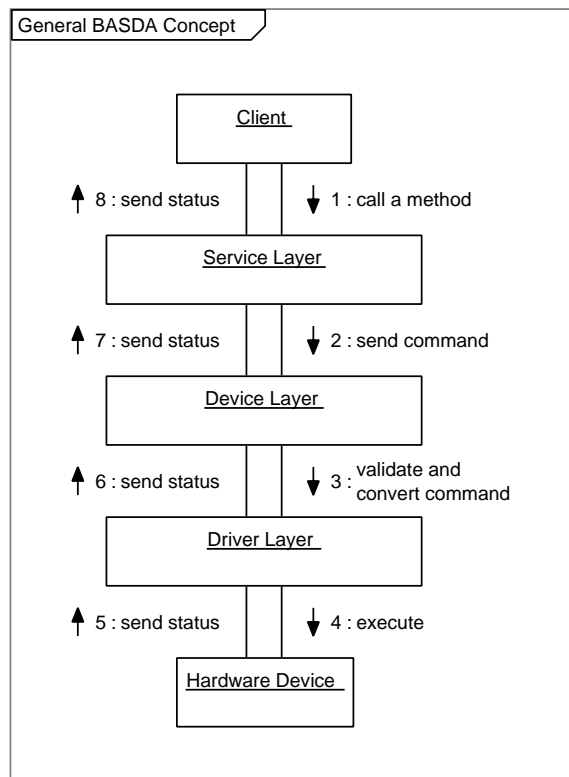


Figure 2. General BASDA Concept. These three software layers form a vertical hierarchy where a higher layer directly controls the next lower layer. The flow of control is strictly downwards, while the flow of the status data is strictly upwards.

The basic concept of the BASDA architecture is to split the package into three layers, which form a vertical hierarchy (Figure 2).

Service Layer The Service part is located in the top layer of the hierarchy and makes the hardware independent high level interfaces remotely accessible. The commands called from the ICS are passed to the middle layer including the Device part.

Device Layer In the Device part the generic interfaces methods are implemented, which validate the command arguments received from the Service layer and calls the hardware dependent methods of the Driver part.

Driver Layer The Driver part is placed in the lowest layer and provides the direct hardware access. The routines of this layer call the hardware device commands directly. This layer is the only part in the concept that is really hardware dependent. It maps the hardware device functionality one to one in order to have for every hardware device feature one elementary method. These methods support only the communication between the Device layer and the hardware devices. No validation of parameter values or parameter ranges are done in the Driver layer. It has to be done in the Device layer.

2.2. Server Architecture

The BASDA software package follows the service oriented approach. It focuses on the necessary Services that are required from the ICS services and eclipses the underlying hardware. A Server consists of several Services, which are loosely coupled into one application. The Services can run on distributed computers in the network and setup a computer spanning Server. The advantage is that the complexity of the entire software can be broken down into simple, small Services, which can be connected together to a powerful Server. It can be configured with different Services as it is necessary, and therefore, it is still understandable and controllable.

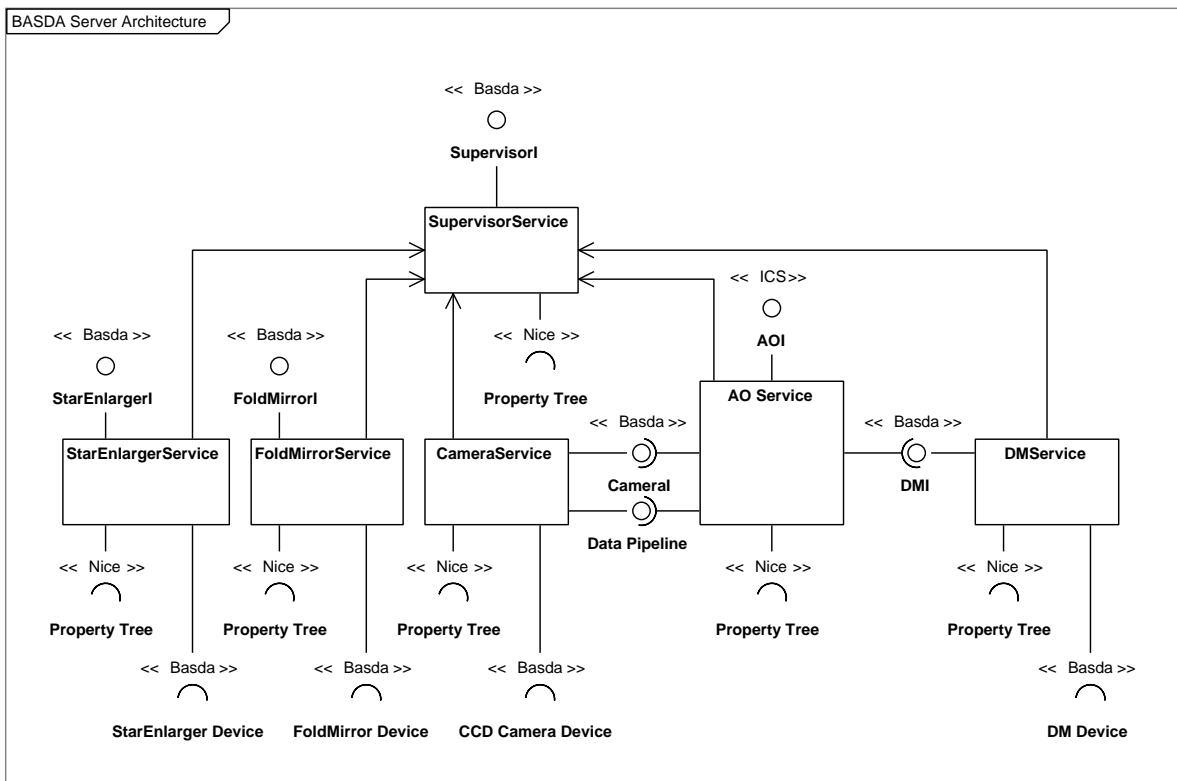


Figure 3. The MHWS Server Architecture. After the AO Service has received the connection address of the Camera Service and the DM Service from the Supervisor Service, it starts driving the DM based on the Camera Service data stream. The AO server is placed in the Sub-System Layer and controls the BASDA Services.

Figure 3 presents the architecture of a BASDA server. In this case it is the design of the MHWS Server, which provides Services for the Star Enlargers, Fold Mirror, Camera and for the DM. All these Services are registered at the Supervisor Service. The Supervisor Service keeps the connection address of each registered Service and

returns it to the connected clients like the Adaptive Optics (AO) Service. The AO Service is part of the ICS Sub-System and controls the BASDA Camera Service as well as the DM Service. The Camera Service pushes the image stream toward the AO Service, where the images are analysed and processed. The resulting signal is sent to the DM Service. Before the AO Service closes the loop to reduce the distortion of the mid-high layer wavefront, the Star Enlargers have to be placed to the reference star positions in the field of view. Therefore, the Fold Mirror Service is moved to redirect the incoming light beam to the Patrol Camera. On the basis of the Camera Service images, the reference star positions are located and the Star Enlargers can be placed to the coordinates of the corresponding reference stars using the Star Enlarger Service.

2.3. Service Architecture

Basically, the Service architecture of each Service is the same. A Service is based on the template based SOA developer framework for astronomical instruments - NICE.⁷ It provides an interface to the upper layer, using C++ interfaces of the Devices and of the Property Tree, and has to manage its own states.

Client Interface Each Service is using the middleware Internet Communications Engine⁸ (ICE) to communicate with the clients. The clients can be either BASDA engineering GUIs, IDL⁹ scripts for laboratory tests, or ICS services. This interface does not have any hard realtime requirements and will be used only for configuration and controlling parameters, and not for data streaming.

Device Interface A Service is configurable by a plain text configuration file, which defines which Device is controlled by the Service using a C++ interface. A Device combines several hardware devices using the corresponding Driver to setup a more complex device. This allows to control more than one hardware devices at once. The Device has to take care of how these hardware devices have to be managed in terms of in what order the driver commands must be sent to prevent damage. A Driver is, as described in the previous section, an API for the hardware access.

Property Tree The Property Tree is a tree structure determined during the initialisation phase of the Service that keeps several status values of the hardware devices and the Service itself. Everytime when a client establishes a connection to a Service it gets a copy of the Property Tree as the Service to which a connection was created. With the help of the Property Tree is it possible to synchronise the connected clients easily. A more detailed description about the Property Tree can be found in (⁷).

Data Pipeline Interface Beside the client interface, some Services provide also a Data Pipeline interface. This C++ interface supports a fast data connection between client and server, and allows, for instance a rapid image stream of a CCD camera.

States The Services provide four different states: OFFLINE, STANDBY, ONLINE and WORKING. The Service is in OFFLINE state after starting the Service application. It is not defined who changes the states of a Service. This has to be define for each Service separate. One case could be that the Service can change the states automatically. Another case can be that a connected client sets the Service in the necessary state. Nevertheless, the Service is in STANDBY state if the Service is initialised. Again, for each Service must be defined what has to be done for the initialisation to reach the STANDBY state. In the STANDBY state the Service is waiting until it is activated from a client. The Service will be changed from the state STANDBY to ONLINE. The Service accept commands coming from the clients only in the ONLINE state. For the time where the Service executes a command, it changes the state from ONLINE to WORKING state (Figure 4).

Figure 5 presents a XY Service, which controls 8 XY Devices. Each Device controlles 2 motor devices using a motor controller driver. Like the Device, which has take to care of how to control the multiple hardware devices, the Service has to take care of how to manage the different Devices. The Service provides the XY interface to the clients to control up to 8 XY Devices. The different connected clients are synchronised via the Property Tree.

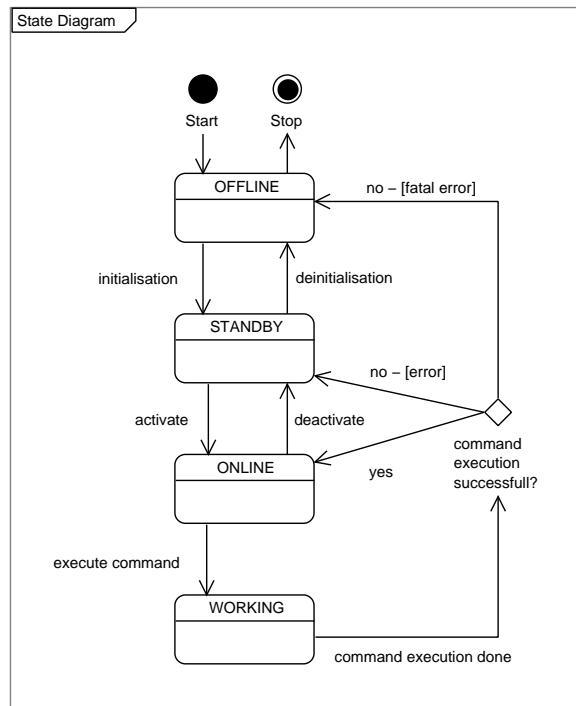


Figure 4. The State diagram. Through the initialisation of the Service, the states are changed from OFFLINE to ONLINE. The Service has to be activated to execute commands. The Service switches from ONLINE to WORKING if a command is executed. Based on the success of the command execution, the Service is changed to the corresponding state.

Generally, the Service itself has to manage the incoming connection requests and has to reject client connections if the controlling of a Device is limited by a certain amount of client connections. For example some devices require that no more than one client has to control that device. This limitation of the connection is defined in the configuration file of the Service.

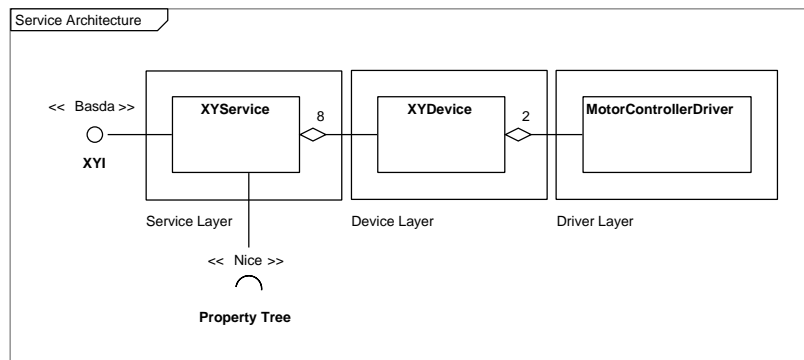


Figure 5. The Service for the 8 Star Enlargers with 2 motor devices each. The BASDA concept with the three layers (Service layer, Device layer, Driver layer) is also readily identifiable.

2.4. Device Interface Architecture

The basic principle of the encapsulation of the hardware dependent low level interfaces from the sub-system dependent services are Generic Device Interfaces. These interfaces establish a base for all Device interface implementations and specify what functionality a hardware independent high level Device interface has to provide. The Service uses these Generic Device Interfaces, and supports thereby different hardware Devices without changing the service itself.

The different hardware devices used in the LINC-NIRVANA instrument are classified and categorised in different groups. For the moment, there are four groups of devices available:

Motor Devices The Motor Devices group consist of all motorised hardware devices. For example linear motors like piezo stages or rotation motors like wheels.

Camera Devices Different camera devices like infrared cameras or webcams are members of the Camera Devices as the name implies.

I/O Devices I/O Devices are devices, which can only be switched on or off. For instance pumps or power management of the instrument belong to that group.

Monitor Devices Monitor Devices are telemetry devices like temperature sensors or pressure sensors, which keep track of data.

The essential characteristics of each group regulate the associated generic interface. The common functionalities of all groups are summarised in the `GenericDevice` interface (Figure 6) and provides abstract methods like `open`, `close`, `init`, `deinit` and `reset`.

```
interface GenericDevice
{
    void open();
    void close();
    void init();
    void deinit();
    void reset();
}
```

- **open, close** The method declarations `open` and `close` are foreseen if special techniques are required to connect to the hardware device and to disconnect from the hardware device.
- **init, deinit** The interfaces `init` and `deinit` are reserved for the hardware initialisation and deinitialisation methods.
- **reset** The implementation of the `reset` interface defines how the Driver and / or the hardware device can be set back.

All Generic Device Interfaces inherit from the `GenericDevice`. In case an additional Generic Device Interface for a new device group is added, the `GenericDevice` can be simply extended by creating a new Generic Device Interface. The following example of a motor device will be used to explain how a generic interface may look like. The generic interface of a motor device contains common motor commands like: `moveToHome`, `moveToReference`, `moveToPositiveLimitSwitch`, `moveToNegativeLimitSwitch`, `stop` and `halt`.

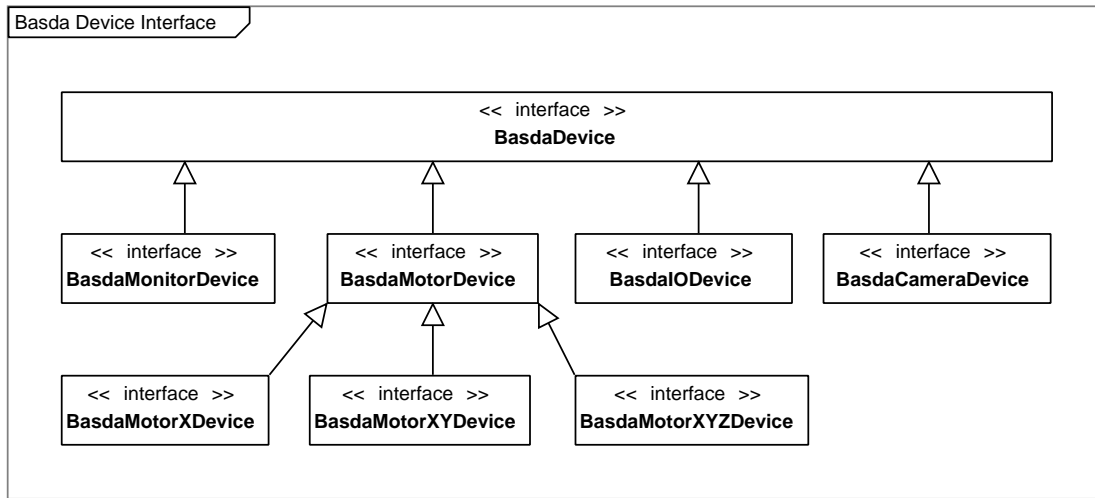


Figure 6. Class hierarchy of the BASDA Device Interface.

```

interface GenericMotorDevice
{
    void moveToHome();
    void moveToReference();
    void moveToPositiveLimitSwitch();
    void moveToNegativeLimitSwitch();
    void stop();
    void halt();
}
  
```

- `moveToHome`, `moveToReference` covers the possibility to move the motor to a defined position.
- `moveToPositiveLimitSwitch`, `moveToNegativeLimitSwitch` moves the motor to one of the motor limit switches.
- `stop`, `halt` can be used, for example, to stop the motor with and without deceleration.

This interface architecture offers a further possibility, the extension of the Generic Device Interfaces with more specific generic interfaces. Figure 6 presents three different kinds of generic motor device interfaces, that differ only in the degree of freedom, because the interfaces provide methods with one, two and three parameters. Depending on the implementation these interfaces can also support devices that require positioning parameters in different kind of units. For example, the interfaces `moveToAbsolutePosition` and `moveToRelativePosition` are able to control two motor devices. Thereby the value of the X position move a length and the value of the Y position an angle. The interface `getPosition` sends back the position values as implemented.

```

interface GenericMotorXYDevice
{
    void moveToAbsolutePosition(double _positionX, double _positionY);
    void moveToRelativePosition(double _positionX, double _positionY);
    std::vector<double> getPosition();
}
  
```

3. APPLICATIONS

In the LINC-NIRVANA software will exist several engineering interfaces on each layer. Engineering interfaces are small applications that allow to test the functionality of software and hardware. The lower the engineering interface is placed in the software hierarchy, the less high level software components are used. Thus, it is possible to locate the failure and helps to debug the software and the hardware easily.

BASDA engineering interfaces are low level engineering interfaces in the LINC-NIRVANA software. Currently, the BASDA engineering interfaces are used in a laboratory of the Max-Planck Institute for Astronomy (MPIA) in Heidelberg and in the laboratory of our team member: INAF - Osservatorio Astronomico di Bologna - to integrate and test hardware components. Two examples are briefly presented - on the one hand an engineering GUI (Figure 7), and on the other hand the CARMA application (Figure 8). Another engineering GUI is the component based astronomical visualisation tool called B5, which is further described in ⁽¹⁰⁾.

3.1. Engineering GUI

The engineering GUI, shown in Figure 7, is written in C++ using QT4^{11,12}. It allows to controll up to 8 single motor devices at the same time by providing the `GenericMotorXDevice` interface with the methods: move to home, move to a relative position (allows to move a motor to the indicated position value corresponding to the current position), move to absolute position, stop and abort (like the halt method described above).

The logging window is in the bottom part of the GUI. It prints all status messages and error messages that come from the hardware.

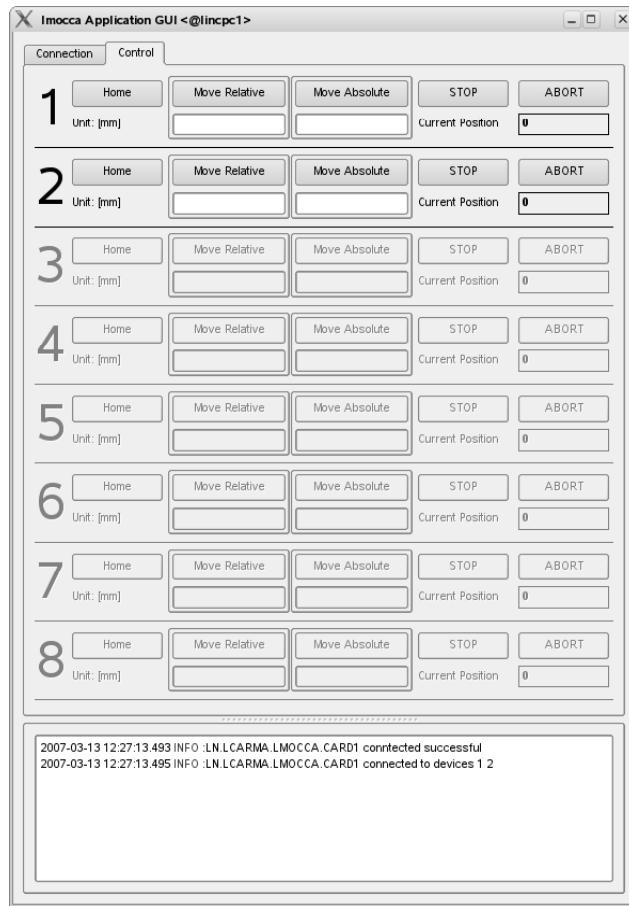


Figure 7. Engineering GUI for motor devices.

3.2. CARMA

CARMA is an application that allows astronomers to do laboratory tests easily. CARMA is based on BASDA and written in IDL, and thereby understandable for astronomers. It is easy to adjust to the upcoming test and supports a simple analysis of the measured data. CARMA controls the hardware of the Adaptive Optics for LINC-NIRVANA, and provides methods to read the images from the wavefront sensor, to send commands to the deformable mirror, to move the motors in the system, to do calibration and closed-loop calculations, as well as some debugging functions and engineering GUIs for the individual components. Currently, CARMA is being used for the integration of the Mid-High Wavefront Sensor to the LINC-NIRVANA post-focal relay¹³ in the laboratory of the MPIA. This integration test validates one single arm warm optics with the Mid-High Wavefront Sensor in close loop in order to ensure the performance of each single element in the final configuration, to check the alignment strategy, the software, and the electronics functionality.

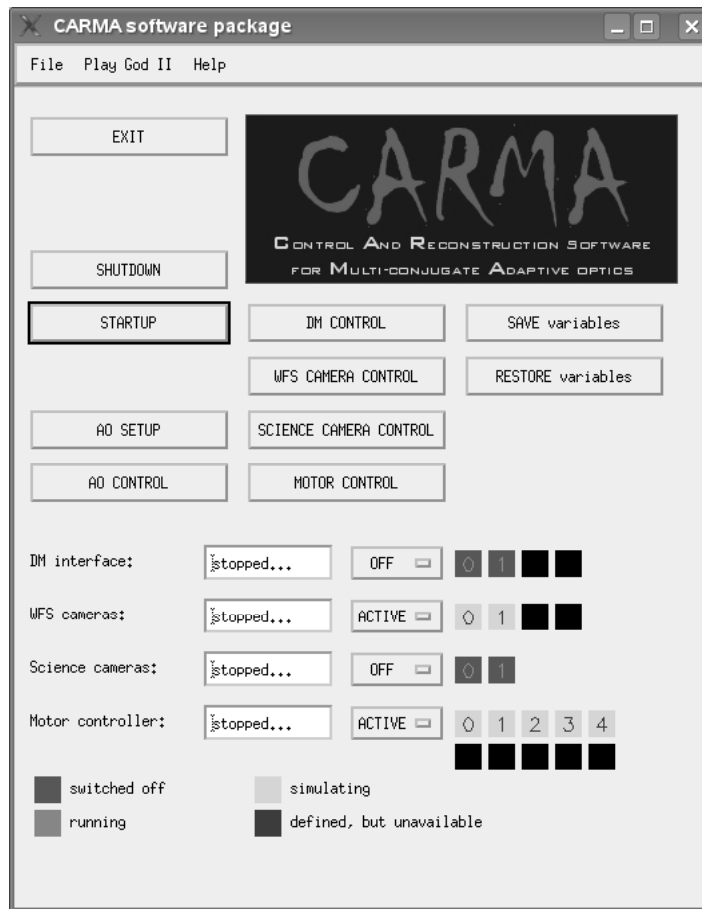


Figure 8. CARMA main GUI.

4. CONCLUSION

We presented a software package that allows to control different LINC-NIRVANA hardware devices remotely. The BASDA package supports different Services that isolate the hardware dependent low level interfaces and provides hardware independent high level interfaces to the ICS. This kind of software design is based on the service-oriented architecture, which keeps the software flexible for possible hardware changes in the future and extendable to support new added hardware easily.

5. ACKNOWLEDGMENT

We like to thank our Italian team members Matteo Lombini and Italo Foppiani, and especially Laura Schreiber (INAF - Osservatorio Astronomico di Bologna) for testing the software in a final configuration of one optical path of the LINC-NIRVANA instrument.

REFERENCES

1. Herbst, T. M., Ragazzoni, R., Eckart, A., and Weigelt, G., “The LINC-NIRVANA interferometric imager for the Large Binocular Telescope,” in [*Ground-based Instrumentation for Astronomy. Edited by Alan F. M. Moorwood and Iye Masanori. Proceedings of the SPIE, Volume 5492, pp. 1045-1052 (2004).*], Moorwood, A. F. M. and Iye, M., eds., *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference 5492*, 1045–1052 (Sept. 2004).
2. Hill, J. M. and Salinari, P., “The Large Binocular Telescope project,” in [*Ground-based Telescopes. Edited by Oschmann, Jacobus M., Jr. Proceedings of the SPIE, Volume 5489, pp. 603-614 (2004).*], Oschmann, Jr., J. M., ed., *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference 5489*, 603–614 (Oct. 2004).
3. Riccardi, A., Brusa, G., Xompero, M., Zanotti, D., Del Vecchio, C., Salinari, P., Ranfagni, P., Gallieni, D., Biasi, R., Andrighettoni, M., Miller, S., and Mantegazza, P., “The adaptive secondary mirrors for the Large Binocular Telescope: a progress report,” in [*Advancements in Adaptive Optics. Edited by Domenico B. Calia, Brent L. Ellerbroek, and Roberto Ragazzoni. Proceedings of the SPIE, Volume 5490, pp. 1564-1571 (2004).*], Bonaccini Calia, D., Ellerbroek, B. L., and Ragazzoni, R., eds., *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference 5490*, 1564–1571 (Oct. 2004).
4. Storm, J., Seifert, W., Bauer, S.-M., Dionies, F., Hanschur, U., Hill, J. M., Moestl, G., Salinari, P., Varava, W., and Zinnecker, H., “Wavefront sensing and guiding units for the Large Binocular Telescope,” in [*Proc. SPIE Vol. 4007, p. 461-469, Adaptive Optical Systems Technology, Peter L. Wizinowich; Ed.*], Wizinowich, P. L., ed., *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference 4007*, 461–469 (July 2000).
5. Xinetics Inc. (Devens, Massachusetts, U.S.A.), www.xinetics.com.
6. Kittmann, F., Gässler, W., Briegel, F., and Berwein, J., “LINC-NIRVANA Instrument Control Software,” in [*Astronomical Data Analysis Software and Systems XVI*], Shaw, R. A., Hill, F., and Bell, D. J., eds., *Astronomical Society of the Pacific Conference Series 376*, 661–+ (Oct. 2007).
7. Berwein, J., Briegel, F., Gässler, W., and Kittmann, F., “A soa developer framework for astronomical instrument control software,” SPIE. 2008.
8. Henning, M. and Spruiell, M., “Ice - internet communications engine.” <http://www.zeroc.com>.
9. ITT Visual Information Solutions, “Idl the data visualization & analysis platform.” <http://ittvis.com/idl/>.
10. Briegel, F., Berwein, J., Kittmann, F., and Alexej, P., “A component based astronomical visualization tool for instrument control,” SPIE. 2008.
11. Trolltech, “Qt sets the standard for high-performance, cross-platform application development.” <http://www.trolltech.com/>.
12. Rathmann, U., “2d plotting widget and more.” <http://sourceforge.net/projects/qwt>.
13. Schreiber, L., Lombini, M., Foppiani, I., Mechke, D., De Bonis, F., Bizenberger, P., Bregolib, G., Cosentino, G., Diolaiti, E., Egner, S., Farinato, J., Gaessler, W., Herbst, T., Innocenti, G., Kittmann, F., Mohr, L., Ragazzoni, R., and Rohloff, R., “Integration of the mid-high wavefront sensor to the linc-nirvana post-focal relay,” SPIE. 2008.