LUCIFER VR: a virtual instrument for the LBT

Kai L. Polsterer*, Marcus Jütte*, Volker Knierim*, Michael Lehmitz*, Holger Mandel*

*aAstronomisches Institut der Ruhr-Univ. Bochum, Universitätsstr. 150, 44780 Bochum, Germany;
*bMax-Planck Institut für Astronomie, Königstuhl 17, 69117 Heidelberg, Germany;
*cLandessternwarte Heidelberg, Königstuhl 12, 69117 Heidelberg, Germany

ABSTRACT

Lucifer VR is a virtually realized instrument that was build in order to allow improved pre-integration software tests, training of observers as well as providing educational access. Beside testing the instrument hardware in combination with e.g. a telescope simulator, software tests need to be done. A virtual instrument closes the gap between regression tests and testing the control software with the integrated instrument. Lucifer VR allows much earlier tests and reduces the amount of time needed to combine the software with the hardware. By modeling the instrument in a simulator, motion times can be calculated very easily and the position of all instrument units can be traced. Especially when using complex mechanisms like a MOS unit a virtual instrument makes software development less time consuming. Lucifer VR consists of three parts; one for handling the communication, another to simulate the hardware and finally a part to visualize the whole instrument in three dimensions.

Keywords: LBT, Instrumentation, Infrared, Spectrograph, MOS, Simulator, Control Software, Software Test Methods

1. INTRODUCTION

LUCIFER (LBT NIR spectroscopic Utility with Camera and Integral- Field Unit for Extragalactic Research) is the near-infrared spectrograph and imager for the Large Binocular Telescope (LBT). It is built by a consortium of five German institutes, Landessternwarte Heidelberg (LSW), Max Planck Institut für Astronomie in Heidelberg (MPIA), Max Planck Institut für Extraterrestrische Physik in Garching (MPE), Fachhochschule für Technik und Gestaltung in Mannheim (FHTG) and Astronomisches Institut der Ruhr-Universität Bochum (AIRUB). It is a full cryogenic near-infrared (NIR) spectrograph and imager (0.9 - 2.5 µm, zJHK-band), equipped with three exchangeable cameras for imaging and spectroscopy, two optimized for seeing limited conditions and the third camera for the diffraction limited case with the LBT adaptive secondary mirror working. Six observing modes are available: seeing and diffraction limited imaging, seeing and diffraction limited longslit spectroscopy and seeing and diffraction limited multi-object spectroscopy. The field of view (FOV) will be 4 x 4 arcmin² in the seeing limited mode and 0.5 x 0.5 arcmin² in diffraction limited mode. For the long slit and multi object spectroscopy (MOS) up to 33 exchangeable masks over the full field of view will be available. The instrument will be equipped with a Rockwell HAWAII-2 HgCdTe-array with a pixel size of 18µm. First light with LUCIFER is planned for autumn 2007. An identical instrument for the second LBT mirror will follow one year later. For more details see [1], [2], [3] and [6].

Due to the complexity of the LUCIFER and in order to fulfill the requirements of the astronomers using this instrument, an appropriate control software is needed. We built a distributed system allowing us to start and stop services without restarting the complete control software. By using Java as the main programming language we could concentrate on solving the tasks instead of spending time on the implementation details (see [7]). Java’s remote method invocation (RMI) provides a powerful framework to write distributed and service oriented solutions. We extended that framework in order to allow services to be started automatically on demand, making our software robust against services failing or hanging up. In order to reduce the complexity of the control software it has been divided into four layers (see fig. 1.). Each layer is responsible for solving the tasks of the given problem domain and forwarding lower level problems to the next layer. This results in having a clear control path from the highest layer to the lowest one allowing us to determine the position of a problem much easier. The lowest layer, called System layer, provides the services needed to persistently store and rebuild data (e.g. to allow to have a central configuration service). The System layer also contains a framework to allow inter-service communication in order to reflect the system state. Other frameworks for internationalization, class documentation, service interaction and service managing can be found in this layer. The next layer is the Control layer

*E-mail: polsterer@astro.rub.de; phone +49 (0)234/3226085; fax +49 (0)234/3214169; www.astro.rub.de
responsible for direct hardware interaction. It contains the services directly communicating with our control electronics and environment monitoring devices. This communication is done by using a RS-232 to TCP/IP mapping port server. The services responsible for telescope interaction and detector readout are also included in the Control layer. As a central element of our software the Journalizer tracks the status of the instrument. All environment monitoring devices and parts of the instrument notify the Journalizer when the instrument state is changing. This service, as a part of the Control layer, reflects the instrument at any time and allows the user to see what is going on. In the next layer the operate logic of the moveable parts inside of LUCIFER are arranged as separate services. The services of the Instrument layer send the movement commands to the motion control services in the underlying layer and notify the Journalizer when the state of an instrument element has changed. In the top layer the instrument is finally operated. A central scheduling service receives observation tasks that contain information on how to setup the instrument, control the telescope and how to make an integration.

2. SOFTWARE TEST METHODS

A problem adapted and easy to maintain software design, a distinct coding style, good and sufficient comments are important for a successful software project. Beside this, software tests should be an elemental part of every software development process. Tests are necessary to ensure that a software complies with its specifications. Unwanted behavior and malfunctions can only be removed by using appropriate test procedures. The final test of course should be the use of the software. In this state the number of errors should be as low as possible. The removal of failures or faults, which are being detected in a late phase of the software creation process, cost much more manpower than to remove early detected errors. Nowadays the test driven development is a solution to deal with this problem. Before starting to write a line of code, the tests to ensure the specification are written. This concept spends a lot of time on defining the tests and helps to provide the user with a software that contains less errors. Apart form the tests that can be done before getting into the implementation phase (e.g. specification tests), regression testing should play an important role. These tests are necessary to ensure that changes to the existing source code, either to remove bugs or to add new functionalities, do not affect the function of other depending code. With the new concept of object oriented software also new methods for software development came up. By using new Integrated Development Environments (IDE) a lot of the tasks a programmer had to do manually could now be done automatically or at least with guidance from the IDE. The LUCIFER control software package is written by using the Eclipse IDE. By adding plugins we could modify Eclipse to allow for RMI-registry and activation daemon interaction, to access the database and ensure correct formatting of the source. The possibility of code completion as well as the syntax and error highlighting reduces the amount of errors occurring during program creation. The majority of errors that had to be removed are runtime errors. We use JUnit tests for automatic regression testing. To run these tests Eclipse provides included graphical functions. Regression testing requires an automated build process that in our case is realized by using ANT. This platform independent tool, which is part of the Apache project, allows not only to build projects. It also provides possibilities to build the documentation, create an archive for software distribution and of course run the JUnit tests. The history of the development process of the LUCIFER control software and especially how it evolved to the use of present day development environments is described in [5].

When writing a control software for an astronomical instrument, the amount of time for completing this task is always an important issue. An often asked question is why to spend time on tests. Tests are done by everyone during the analysis and design phase of a software. In this phase a lot of time is spent on defining the functionalities that are needed by the software, and by using examples these use cases are tested on completeness. Without this planning bigger project will necessarily discover a lot of problems and errors – a missing function that is needed can be considered as an error – and therefore do not need to use tests. Regression tests are nothing more than the tests that will necessarily be written in order to test if the software is working. The test values printed to the command line can be easily transformed into JUnit test cases that can be rerun automatically after every change to the source code. These tests will save the same amount of time on error removal that in the beginning was spent in test creation.

As mentioned before the resource time is often insufficient when writing software. In nearly every case where the software should control a piece of hardware the same discrepancy can be discovered. Hardware development is finished and everyone expects a fully functional software. The main problem is that tests of the software – hardware interaction could first start when the hardware is available. Of course the software could be created parallel to the hardware but the time needed to complete the software for a part just being finished is not negligible. In order to get out of this dilemma we decided to build a virtual instrument that we could use for our software tests and thus reduce the amount of time necessary to adopt the software to the real hardware. This virtual instrument allows the test of the movement logic in a pre-instrument integration phase and reveals elemental errors at an early moment of the software development process.
Fig. 1: LUCIFER control software package extended by services needed to provide functionality of a virtual instrument.
3. LUCIFER VR

Lucifer VR is the virtual realization of the LUCIFER instrument. It was build to provide an interface to the instrument hardware in order to run the instrument control software without a real instrument. A first prototype of this virtual instrument is described in [4]. Cause to the layered architecture of the LUCIFER control software the simulator could easily be added. The design of the control software uses the Control layer to interface with the control electronics. By just replacing the hardware interfaces with the simulated ones a virtual instrument is realized. On details how the simulator is integrated into the control software see fig. 1. When talking about the different hardware interfaces that have to be replaced by simulated ones one has to distinguish between these used for environment tracing and those used to interact with the instrument parts itself. The software packages used for readout and telescope interaction do not provide the possibility to be simulated through an exchanged hardware interface and therefore need a special treatment.

3.1 Environment

In order to simulate the environment mapping monitors a service called ValueStorage was added. This service is responsible for storing the state of a monitor. This means it stores e.g. parameters that have been sent to a monitor and may be queried later. It also provides the functionality of providing random numbers. Therefore a bunch of random distributions exists that can be used to describe e.g. a temperature behavior of an element in connection with the current simulation time. By realizing the ValueStorage as a central service the state of the environment simulation can be controlled very easily. The DataDisplay belonging to the Operation layer allows direct access to these values controlling the state of the different monitors. These values can be changed while the software is running. After the next query is send to a simulated monitor these modified values are used. Cause to the requirements of the environment monitoring and environment controlling electronics a simple and centralized storage is adequate to realize complete monitor function without existing hardware.

3.2 Readout and Telescope

One of the more complex parts is the simulation of the readout and telescope interaction, because these software packages are provided from third party and do not foresee the use of a simulator. For the telescope interaction we are looking forward to get a telescope simulator software that is currently being planned by the LBT software group. The simulation of the readout process is done by using the Astrophysical Virtual Observatory (AVO) to create imaging data. The influence of the instrument on the data like filter transmission curves or dispersion are neglected. Until now no special image treatment is done but distortion, noise, sky or other artifacts may be added later. For the spectroscopic mode no data is available. The simulation of the virtual instrument does concentrate on the interface between the software and the hardware. It is not simulating the optical path or any other physical factor but, in order to provide a correct interface to the control software, the simulation needs to cover the aspect of the physical motion of the inner parts.

3.3 Instrument

To understand the realization of an appropriate simulation of the mechanical parts in the real LUCIFER instrument two aspects are important to account. First the rebuild of the electronic communication interface and second the simulation of the mechanical movement. In projects where software has to control electronics a command specification exists, defining the way how to interact. In case of the LUCIFER instrument this interface is given by a numerical command language. The electronic communication was modeled by using a two way parser. This parser was designed to analyze and create commands as well as the matching responses. This enables the parser to be used for the electronic controlling software services “Instrument Motion Control Unit” and “MOS Motion Control Unit” which are both addressing a separate electronics box. The amount of time spent to create this parser is therefore negligible. The two way usage of the parser leads to simple tests of the communication interface used for addressing the electronics. Before writing any line of code that interacts with the electronics, the specification of the allowed commands can be used to test if the parser could communicate with another instance of itself. After this tests the parser could be used to test the communication with the electronics. In the LUCIFER project these tests revealed some errors in the communication protocol where the real communication differs from its specification. This parser is one part of the communication interface replica.
The other part is an emulation of the control electronics firmware that is responsible for connecting the communication interface to the simulation of the mechanical instrument parts. In this firmware emulation the time needed for command processing and response creation is taken into account. Incoming commands are transferred into tasks processed by the simulator. During task creation events may also be added to the event handler of the simulator. This may for example be the event of a motion being completed or a limit switch that is reached by a motor. The hitting of a limit switch results in an error response that needs to be created. When tasks are added to the simulator the model time an event will occur can be previously calculated and therefore be scheduled for the right moment. These events can then be used by the firmware emulation in order to create the matching electronic responses.

The simulation model of the LUCIFER instrument uses two simulators for every motor. One simulator to simulate the real motion and the other one to rebuild the internal motor impulse counting. This is necessary to model the behavior of a stepping motor being driven with a speed or torque out of its physical specifications. In this case random functions are injected in order to modify the real motor position. See the motor velocity display in fig. 4 on how this modification is visualized. Hardware tests carried out show that the behavior of a stepping motor used beyond specified limits is unpredictable. Random functions are used to rebuild this by letting the real value differ from the expected on. Of course this modeling approach does not rebuild the real instrument behavior but it allows us to find errors in the movement logic. All the simulators used for the motors use the same model time generator that provides the function to have the model and the real time run synchronously.

![Class diagram of the simulation framework.](image-url)

Fig. 2: Class diagram of the simulation framework.
The Simulator

The simulation framework represents the central element of the virtual instrument. Besides the ValueStorage used to rebuild the environment monitors and controllers, this framework allows to use the control software stand alone. When designing this framework (see fig. 2) the prototype described in [4] was taken and modified in order to be more flexible. The simulation framework is embedded in a distributed environment allowing remote access in order to observe or modify a simulation. To allow these modifications or observations, a remote callable implementation of the Observable and Modifiable interface is connected to each simulator. By distinguishing between observers and modifiers, the behavior of a simulated element could be easily modeled. The main function of the simulator is to manage the SimulatorTasks that are generated by the firmware rebuilding layer. If the current value that is simulated is asked, the simulator processes the queue of tasks up to the current model time that is taken from the central TimeGenerator. This processing is done by solving the mathematical function that is included in every task.

The observers and modifiers that are registered to a simulator will be informed about the new task. Modifiers have the possibility to fold the function embedded in this SimulatorTasks with other function. This is e.g. useful to add some noise to a motion when driving a stepping motor too fast. Observers are used to track the state of the simulation. In case a simulation function analyzed by an Observer will reach a given value within the parameter space the function is defined, events could be created in order to respond to the firmware emulating layer. Connected to the simulators are observers that track whether a movable part of the instrument is an area where a switch is enabled. This is e.g. the case when a limit switch is hit and the motion, to be precise the simulator, needs to be stopped. On this way the SwitchSimulator, that interacts with the electronics used to retrieve switch information, is implemented.

By using the Observer interface the visualizing component could directly connect to the simulator. The three dimensional display is enabled to calculate the motion path on its own, just by analyzing the simulation function. In case the simulated value is changed or the simulator is stopped, all registered observers are notified. The simulator is used to represent the numerical part of the simulation framework.

The event-based part of the simulation is realized by the TimeGenerator. This important element is responsible for scheduling events. Therefore this service contains a queue that is sorted by the model time. By linking the real system time with the model time the simulation looks like being an instrument running in real time. The TimeGenerator contains functions to modify the coupling factor between the real and the model time in order to run the virtual instrument in slow motion.

By using this flexible framework everything, that can be described by a function, can be simulated. As described above, in case of a simple stepping motor, two simulators are used. Today the virtual instrument just simulates the motion of the inner parts although other parameters could be modeled using this simulation framework.

4 TESTS WITH LUCIFER VR

After the virtual instrument was built, several ideas on how to use it arose. The virtual instrument is mainly used for three tasks. These tasks are important to the software development process and have revealed a lot of errors until now. First of all the tests of the services responsible for direct electronic interaction have to be mentioned. The virtual instrument allows stress testing of this services without wearing down the parts that are later used in the instrument. During these tests, a lot of errors concerning the simultaneous operation of multiple motors, could be revealed. In a typical case of instrument usage these errors may have occurred indeterminately and may have lead to failures that could hardly be traced. The thread synchronization bugs, that are responsible for these errors, could be fixed before using the real instrument. Another benefit, concerning the services used to communicate with the electronics, is to be able to calculate the time that is needed for a motion with respect to all motor parameter and the distance to drive. This analytical modeling of the stepping motor is used to calculate exact motion times allowing the creation of timeout exceptions in case the electronics do not confirm a motion to be completed in time. This solution allows noticing communication errors between the hardware and software in a low layer of the control software providing the possibility of counteractive measures instead of continuing to send commands to the electronics.

The second important point on using the virtual instrument is that we could entirely test the movement logic of the Instrument layer. Without Lucifer VR the tests and the programming of the Instrument layer could only start when a part controlled by one of these services is finished. Now using the virtual instrument these tests could be independently been carried out. These tests help to define the states from where a part transits into an other. After defining this motion logic
The last important point is that the virtual instrument allows the test of the complete software before the instrument is entirely integrated. This system tests are important to test the functionality of the Operation layer. Cause to the layered architecture of the LUCIFER control software package all depending layers do not recognize whether they are using the real or the simulated instrument. The exchange of the interface between the instrument controlling electronics and the software is made transparent. With the possibility of a fully functional scheduler service, as the core of the operation layer, problems like observation macro definition could be started to work on. In addition to these tests the graphical user interfaces could also be tested in previous and will show the same timing behavior as when connected to the real instrument. On details concerning the graphical user interaction see [8].

5 OTHER BENEFITS

Beside the testing capabilities provided by the virtual instrument other benefits have to be mentioned. First of all the ability to visualize the instrument in three dimensions. Before parts of the LUCIFER instrument could be visualized the CAD drawings need to be preprocessed. After converting the CAD data to a format that can be displayed by Java3D which is used for visualization, the number of polygons used by the different elements needs to be reduced by 90% in order to allow a fast reacting display. Figure 3 and 4 demonstrate the similarity between a real unit and the visualization of the simulated one. In order to show the motion of a component the display registers – using the Observer interface – to the simulators which represent the motors that drive this instrument part. By using a function describing the translation, rotation and tilt of the moveable parts depending on the position value of the motor, the three dimensional view was realized. This feature allows the engineer to understand what is going on in the LUCIFER cryostat and to track the instrument status in a human friendly way. In case of a malfunction the engineer can use this visualization component, which is integrated into the user interface, to display the last known secure state stored by the Journalizer. Seeing LUCIFER in three dimensions and locking at the different parts interacting enables the user to understand how lucifer is working. Another advantage is that by using the virtual instrument the observer will be able to use the instrument control software prior to the observation visit. This will result in observers being familiar with the graphical user interface and the amount of observation time spent on slow user interaction may be reduced. By using the instrument control software without a real connection to the telescope and instrument the observers may test their complete observation runs, because the time the virtual instrument uses for movement is within milliseconds identical. We expect that a trained visiting observer with a pre-tested observation run will produce better results with LUCIFER than an untrained. The last but probably the most important benefit of the virtual instrument is that we could provide educational access to LUCIFER. This will allow everyone to use the instrument control software like the astronomers do. By getting data from the Astrophysical Virtual Observatory, virtual observation runs can be planned and scheduled. Furthermore the three dimensional view of the LUCIFER instrument will present the mechanical complexity needed to receive scientific data. The virtual instrument will demonstrate to the public how astronomers use a near-infrared instrument in order to gain data for their experiments.
ACKNOWLEDGMENTS

The virtual instrument is funded by the Klaus Tschira Foundation. This work is embedded in a project supported by the German federal department for education and research (BMBF) under the project numbers: 05 AL2VO1/8, 05 AL2EIB/4, 05 AL2EEA/1, 05 AL2PCA/5, 05 AL5VH1/5 and 05 AL5PC1/1.

REFERENCES

[1] W. Seifert et al., LUCIFER: status and results of the hardware testing, SPIE proc. 5492, 2004
[8] V. Knierim, M. Jütte and K. Polsterer, User interaction with the LUCIFER control software, SPIE proc. 6274, 2006

Fig. 4: Comparison between a real test unit and the visualization of the simulator behind a motor velocity display.