User interaction with the LUCIFER control software

Volker Knierim, Marcus Jütte, Kai Polsterer and Jan Schimmelmann
Astronomisches Institut, Ruhr–Universität–Bochum,
Universitätsstr. 150, 44780 Bochum, Germany

ABSTRACT

We present the concept and design of the interaction between users and the LUCIFER Control Software Package. The necessary functionality that must be provided to a user depends on and differs greatly for the different user types (i.e., engineers and observers). While engineers want total control over every service provided by the software system, observers are typically only interested in a fault tolerant and efficient user interface that helps them to carry out their observations in the best possible way during the night. To provide the functionality engineers need, direct access to a service is necessary. This may harbor a possible threat to the instrument in the case of a faulty operation by the engineer, but is the only way to test every unit during integration and commissioning of the instrument, and for service time later on. The observer on the other hand should only have indirect access to the instrument, controlled by an instrument manager service that ensures the necessary safety checks so that no harm can be done to the instrument.

Our design of the user interaction provides such an approach on a level that is transparent to any interaction component regardless of interface type (i.e., textual or graphical). Using the interface and inheritance concepts of the Java Programming Language and its tools to create graphical user interfaces, it is possible to provide the necessary level of flexibility for the different user types on one side, while ensuring maximum reusability of code on the other side.

Keywords: User Interface design, Software concepts, Java, NIR instruments

1. INTRODUCTION

LUCIFER (LBT NIR Spectrograph Utility with Camera and Integral-Field Unit for Extragalactic Research) is a new near-infrared imager and spectrograph, to be used at the Large Binocular Telescope (LBT) on Mt. Graham in Arizona, USA. The covered wavelength range will be from 0.9 to 2.5 microns. It is currently build by a consortium of five german institutes and will be commissioned in 2007. For more information about the instrument see the references 1–3.

The complexity of this instrument requires special care in designing the control software. This is also extremely important when the possible mechanisms of user interaction have to be developed. As different types of users, for instance engineers and observers, require different abstraction levels of the user interface (UI) and different access to the instrument, this has to be implemented also in the software. Engineers need full control over the instrument and therefore the different service available to them. Depending on the operational status of the instrument, i.e., integration in the lab or maintenance on the mountain, engineers want (and need) direct access to basic controls or higher level controls. During lab integration, specifically selecting a single motor and moving it will be necessary, while on the mountain control over the MOS unit may be necessary after mask exchange. There is of course always the thread that the instrument can be damaged, when direct access is provided, but it is the only way to make sure that errors can be found and corrected for by the engineer.

Observers on the other hand, should not have (and do not need) this direct control. They seek an interface that is most suited to their needs. Usually these would be the execution of the intended observations as fast and as efficient as possible, to maximise the usability of, and scientific output from the instrument. They need a UI that is simple to use and easy to understand. Control of the instrument components (camera wheel, grating unit, filter wheels, etc.) needs the supervision of an instrument manager service, that ensures safe operation of the instrument. Therefore, only indirect access to instrument components is allowable.

In this paper we present our software design to provide, and take care of, the requirements mentioned in the last paragraphs. In section 2 we give a brief overview over the complete software package and its layered
structure. Section 3 describes our mechanisms to support different access control levels, while ensuring maximum usability of source code at the same time. The implementation of these mechanisms will be described in detail in section 4. Section 5 presents a first prototype of the graphical user interface developed for the future observers of LUCIFER as well as a description of the observation preparation tool (OPT) that future observers can use to prepare their observations before arriving on the mountain. We summarize our work in section 6.

2. SOFTWARE DESIGN

In this section, we will give a general overview of the LUCIFER control software. We have used the Java programming language to develop our software. For detailed information about the software see references 4, 5.

2.1. Layered Structure of the Software Package

Figure 1 shows the complete LUCIFER control software package. We chose a layered approach for our software design, using four layers representing different abstraction levels of control and functionality. The lowest layer, called the system layer, consists of fundamental services for the software. This includes configuration, time and message services used for control and configuration of the distributed system as well as crucial services like the activation system and registry.

The second layer, called control layer, resembles the services that are necessary to control the instruments. These include the motion control units (MCU’s) for the MOS and the instrument, the environmental control services as well as others.

The third – instrument – layer, is composed of services controlling the different components of the LUCIFER instrument. Every unit of the instrument is controlled by a dedicated service which is using the functions provided by the second layer. Services to control the filter- and camera wheels, the grating unit and others are found in this layer.

The highest layer is called operation layer. In this layer all components necessary for the operation of the instrument are confined. These include the instrument managers for the two LUCIFER instruments (one for each instrument), a scheduler for automatic observations using predefined observation blocks prepared by an user, and the user interfaces components for the observer and engineer. All these components are described in full detail in references 4 and 5.

2.2. User Interaction With the Different Layers

Engineers will obviously need access to all layers of the software system. During integration of the instruments the interaction will mostly be with the services of the control layer, while normal maintenance of the instrument on the mountain will mostly need to use the services provided by the instrument layers. Access to the system and operation layer is of course possible, but will usually be seldom needed.

Observers will only have direct access to the components of the operation layer (i.e., scheduler and instrument managers), which then control the instrument. Therefore, there is only an indirect access to all services of layers one to three for observers.

3. USER INTERACTION PRINCIPLES

Firstly, this section describes on an abstract level, how the different needs for user interaction (see the introduction in section 1) with the software are modeled and realized in our design. After the general principles are introduced, one example is presented that describes in detail how these principles are applied to the software system services. By using the inheritance principles of object-oriented software analysis and design, it is possible to optimize code reutilization and simultaneously providing well readable code.
Figure 1. The LUCIFER control software package
3.1. Access Control Levels and User Interface Constraints

Access control to the software is modeled in our design using three marker interfaces that we call service link interfaces. These are called Basic, Observer and Engineer. The first interface type Basic is used to specify a rudimentary access to the services for data display, and allows no manipulation (either direct or indirect) of the services. The second interface type Observer defines access at the observer level. This will always be indirect access to a service via the controlling instrument manager service. The third type, Engineer, defines full control over a service using a direct connection. The three access control interface are connected via inheritance, in the way that Observer extends Basic and Engineer extends Observer (see Figure 2). The use this inheritance structure establishes an easy way to ensure that Observer has at least the functionality of Basic, while Engineer provides at least the functions of Observer. In other words: An engineer can always do everything that an observer can, but an observer may not have the same possibilities that an engineer has.

The access control interfaces also define constraints on the possible UI. Our design principle is suitable for different types of user interfaces, i.e., textual or graphical. For graphical user interfaces (GUIs), we define an abstract class called LPanel, which is the root class of all GUI components used in the software package. A class diagram of this class is shown in Figure 3. As can be seen in the diagram, three constructor operations are provided by the class, one for each access control interface. Calling one of these constructor defines the type of user interface that is used. Because of the inheritance structure of the access control interfaces, it is possible that an engineer can use an observer type user interface, while the reverse case is not allowed, i.e., an observer never can have an engineer user interface. The operation hasDirectAccess() is used to determine if the user interface is directly connected to a service or not. The access level is also directly visible for the user through the use of different background colors. This is done implicitly by every constructor operation calling the private
3.2. One Example

We will now use one example to describe the process of access control definition in detail. It uses the Motion Control Unit (MCU) service found in the control layer of the software package. The service has an RMI interface defining the functions available for the unit. Listing them all here would greatly exceed the context of this paper. In general one can say, that methods are provided to get the allowed motors, start and stop the motion of motors, open and close magnets, get and set velocities and position of motors, tests for movements of a specific motor, and so on. All operations defined in this interface are split into the three access control interfaces: `MCUBasic`, `MCUObserver` and `MCUEngineer`. Figure 4 shows a class diagram with the inheritance structure. As can be seen in the figure, every access control interface for the MCU service extends the corresponding marker interface, as well as the relevant interface of lower access level (`MCUEngineer` extends `Engineer` and `MCUObserver`). The `MCUBasic` interface only offers methods to get the allowed motors for the MCU and to test if a motor is moving. The `MCUObserver` provides methods to get the position and velocities of motors, among others. Finally, the `MCUEngineer` interface permits the starting and stopping of motors, setting velocities and positions, open and close the magnets, etc. By inheritance, the MCUEngineer has the full functionality of the RMI service, and can use different user interface types for display.

4. IMPLEMENTATION DETAILS

In this section, we will show some details of the implementation used for GUI components. We will use the GUI component for one of the filter wheels of the LUCIFER instrument, to demonstrate the possibility of code reuse. We use a different service now, because the example given in section 3.2 will only have an engineering
GUI, whereas the filter wheel GUI will be used by an observer GUI and an engineer GUI. The different methods to provide access control will be demonstrated using collaboration diagrams.

Figure 5 shows the GUI component that will be used to control the filter wheels. The left GUI shows the interface as it will be displayed for an engineer, while the right GUI is the observers view of the GUI. The background color of the engineer GUI differs from the observer view to indicate clearly that the access to the service is direct. Otherwise, the GUI components look the same and are indeed created by the same class. For the left GUI display, an constructor operation with an $FW_{Engineer}$ interface was used, for the right GUI the constructor operation with an $FW_{Observer}$ interface was used.

To explain in detail, how the access to the service is managed, Fig. 6 and 7 show, how the different implementation classes provide the required methods. Figure 6 demonstrates, how the engineer accesses the service. As can be seen in the left diagram, a call to the setFilter operation is directly forwarded to the filter wheel service in layer three of our software. This service uses two services of the control layer: firstly the MCU to move the corresponding motor of the filter wheel, secondly the Journalizer to store the new position of the filter wheel after the movement has finished. By this implementation, the movement is executed as soon as the engineer commands it, without performing any further security checks.

The right diagram shows, how the getFilter operation is implemented. The call is directly forwarded to the service in layer three, which itself returns the current filter position using the HIRAMO service of the control layer.

Figure 7 shows, how this access is done using an observer interface. A call to setFilter is encapsulated into an ObservationBlock (see section 5.2 and Fig. 9) as one instance of an InstrumentSetup. The scheduler passes this setup to the InstrumentManager when it executes the OB. The InstrumentManager performs all necessary security checks and moves the different components in correct order. To do that, it relies on the services provided at the instrument layer of the software. From this point on the actions performed are the same as for the engineer case. When the observer calls the getFilter operation, the implementation calls the Journalizer service to get the last position that was stored for the corresponding filter wheel. In both cases, no direct connection is present between the observer and the lower layers of the software.

5. LUCIFER OBSERVER INTERFACE

After describing the general principles of our software design for the user interaction with the software, we will present a first prototype of an observers graphical user interface in this section. Following this, we will give a brief overview of the Observation Preparation Tool (OPT) that future observers can use to prepare their observations before arriving on the mountain.

5.1. A First GUI Prototype

Figure 8 shows a first prototype of the graphical user interface that will be available for observers. It is structured in three major regions:

1. The telescope control area,
2. the instrument and detector control area and
3. the system message area.

The telescope control region of the user interface is relatively large compared to the user interfaces of other instruments. This is mainly because the design of the LBT Telescope Control System (TCS) requires substantial communication with the instrument that is using it. This includes the positioning and possible offsetting of the mirror, as well as activating the adaptive optics (AO) and tracking modes. The required information can be entered using the corresponding buttons and a subsequent committing of the new telescope setup. If the information is retrieved from the TCS it will displayed in this area.

The instrument and detector control area show all major optical components of the LUCIFER instrument and their current state. The current state and substate of the instrument is shown in the top left, while the top right indicate if the two LUCIFER instruments are used simultaneously.
Figure 6. Collaboration Diagram showing engineer access over the filter wheel service

Figure 7. Collaboration Diagram showing observer access over the filter wheel service
The major components include the multi object spectroscopy (MOS), grating–camera wheel– and filter wheel units. The MOS unit has two separate mask cabinets, one fixed and one movable. The movable cabinet can hold up to 23 masks. Only loaded mask positions are shown in the GUI of the unit, i.e., the image shows the display when 18 masks are loaded in the movable cabinet. The filter wheel shows that it is currently in position “Ks” while the user selected the ‘z’ filter for the new setup of the instrument. This is indicated by the different background color of the corresponding button. The information panel provides access to different status values, like the position of the moveable mirrors inside the instrument, ADC and tip-tilt unit. Access to environment information, i.e., temperatures and pressures inside the instrument is also possible. If the new setup for the instrument has been chosen by the observer, committing the setup will start the configuration change.

Control of the detector is separated from the instrument setup to allow multiple exposures with the same setup on the one hand, as well changing the instrument setup without starting an exposure at the same time. Committing a new detector setup will change the different settings, like detector integration time (DIT), number of DITs (NDIT) and number of integrations (NINT) to the selected values and start the exposure. The last settings for DIT, NDIT and NINT are shown for information as well as the file name of the next exposure. The last region is shown on the bottom and displays messages generated by the software system message service, that are relevant for the observer.

The buttons with a yellow and black border around them immediately terminate the changes that should be made to the separate unit, i.e., telescope, instrument or detector. The optical design is chosen in a way, that they clearly differ from all other buttons and can quickly be found visually. The stop button in the top right part of the interface will stop all subsystems at once in case of an emergency.

The instrument control interface for the LUCIFER instrument will be supported visually by a virtual instrument that is developed simultaneously. It will show animations for all movements that occur in the real instrument. For more information about the virtual instrument see reference 6.

5.2. The Observation Preparation Tool

The Observation Preparation Tool (OPT) is intended to help the observer to prearrange an observation run. The observer can create so-called ObservationBlocks by entering the important information for an observation. The OPT should support the observer in planning the observation run and point out possible errors. In addition the OPT will be able to generate an ideal schedule out of the observation information entered by the observer. Such a schedule can be modified by the observer at any time to meet special conditions.

The ObservationBlock (see Fig. 9) will be the basic data component of an observation. It consists of Tasks which represent the different functions provided by the LUCIFER instrument. A Task could be an ObservationTask, an InitialisationTask or an SelfTestTask. An InitialisationTask will set the instrument in a well defined state, i.e. it will prepare the instrument for oncoming instrument setups. The SelfTestTask will cause the instrument to test itself for possible malfunctions. However, the ObservationTask will be the most used Task because it contains the setups for the telescope, the instrument and the detector. Each Task includes a TaskDescriptor, so the observer can easily find out the task’s purpose without checking the entire task setup.

An observer creates an ObservationTask by specifying the desired TelescopeSetup, InstrumentSetup and DetectorSetup. This could be done by specifying a whole setup containing all three components or just parts of the setup. E.g., by specifying just a ReadoutSetup, the observer is able to make a single integration without changing the current telescope and instrument setup. In order to maintain a certain order of the Tasks the observer can combine several Tasks to an ObservationBlock. By doing so the sequence of the Tasks in that block is fixed and will not be changed by the scheduler. However, the sequence can still be changed manually by the observer during the observation.

During the night, the observer will mainly interact with the scheduler which executes the Tasks in the ObservationBlocks. The scheduler can be stopped, paused and resumed. If the scheduler is paused while executing an ObservationBlock which contains more than one Task, it will save the current state so the ObservationBlock can be resumed from the last fully executed Task in that block. During the whole observation run the observer can rearrange and change the ObservationBlocks which are not yet executed.
Figure 8. The prototype of an observers GUI
Figure 9. Schematic diagram of an ObservationBlock
Since the scheduler consists of two separated observation queues, it provides the functionality for parallel observations with both LUCIFER instruments at the same time. The requirement for parallel observation are matching telescope setups in both ObservationTasks. If this precondition is fulfilled, the scheduler sends the Tasks to both instruments and the TelescopeSetup to the telescope. However, parallel observation can be by-passed if the user specifies which instrument should execute the ObservationTask. After the execution of a Task, it will be stored in a separated queue, so the observer can reschedule already executed ObservationBlocks, in order to repeat failed observations.

6. SUMMARY

We have presented our concept of the interaction of the different kind of users with the LUCIFER instrument control software package. Our design takes care of the different needs depending on user type, i.e., engineer or observer, while ensuring maximum flexibility and reusability of source code at the same time. Using three different access levels – basic, observer and engineer – connected by inheritance, and different interface views for these levels, programming time is minimized. The same code can be used to create variable graphical user interfaces providing different access and security methods.

We have shown the first prototype of a graphical user interface for the future LUCIFER observer as well as the concept of our observation preparation tool (OPT) to create observing blocks prior to the observing runs. This allows for minimum time loss and maximum usability of the instrument due to well prepared observations that can be scheduled to the instrument in an efficient way.

ACKNOWLEDGMENTS

This work is 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