Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

Graphical User Interface for an Observing Control System for the UK Infrared Telescope

M. Tan, A. Bridger, G. Wright

UK Astronomy Technology Centre, Edinburgh, EH9 3HJ, United Kingdom

A. Adamson, M. Currie, F. Economou

Joint Astronomy Centre, Hilo, Hawaii 96720

Abstract. A Graphical user interface for the observing control system of UK Infrared Telescope has been developed as a part of the ORAC (Observatory Reduction and Acquisition Control) Project. We analyzed and designed the system using the Unified Modelling Language (UML) with the CASE tool Rational Rose 98. The system has been implemented in a modular way with Java packages using Swing and RMI. This system is component-based with pluggability. Object orientation concepts and UML notations have been applied throughout the development.

1. Introduction

A new observation control system (OCS) is one of three parts in the ORAC project (Bridger et al. 1998). The observation control system consists of two sections, the observation sequencer and the observation manager (OM). The ORAC OM provides a Graphical User Interface and some related networking for the new OCS.

The development of the ORAC OM includes its analysis, design, implementation and some testing. The software has been developed as a component-based and object oriented system for good reliability and easy maintenance. It has been analyzed and designed by use of the Unified Modelling Language (Booch et al. 1999) with a CASE tool (Rational Rose 98, Quatrani 1998). The implementation is mainly in Java due to the advantages of the language in GUI and networking development. The system has been subject to some levels of testing.

The development process has been intended to follow the basic UML characteristics, i.e. use-case-driven, architecture-centric, iterative and incremental (Jacobson et al. 1999).

2. Analysis

The system analysis is mainly based on capture of the functional requirements of the system to give a sound grouping of these requirements, in order to partition the OM system for implementation.

Internals of the system: In UML, use cases capture the functions of a system. The system provides two basic functions: Graphical User Interfaces and networking. It is obvious that we can divide the functions into two basic use cases. One is the graphical interfaces. The other is the networking e.g. the socket communication with DRAMA (Farrell et al. 1993) tasks. We call the later use case a "DramaSocket".Within the graphical interfaces, there is a clear separation between two kinds of user interfaces. One is the user interface before starting an observation and the other is the user interface during an observation. As a result, the user interfaces are grouped into two use-cases. One is "science program" which is used before an observation. The other is "sequence console" which is used during an observation. In addition, an online help feature is required, therefore the last use case is the "help tool".

Externals of the system: Apart from capturing the functional requirements, use cases are also used to find the system boundary and its external contacts by identifying who interacts with the use cases. In our case, we can find a few external sources (called actors in UML) to use the use cases. They are astronomical observers, drama tasks, telescope support staff and the Observing Tool (Wampler et al. 1997) database server.

After analysis, the design focused on how to realize these use cases by identifying suitable Java objects/classes for implementations. A standard way of using UML to do this is to go through a number of scenarios for each use case to find appropriate objects. More importantly, these scenarios help to identify how these objects interact with each other.

3. Design

System partition: The first step of design is the partition of the system. It is obvious that four sub systems or Java packages are required from the use case analysis. They are the DramaSocket, sequence console, science program and helptool.

Design process: Going through one or more scenarios for each use case. It helps to produce a number of UML sequence diagrams. Sequence diagrams are used to produce class diagrams.

A class diagram shows the static structure of a system in terms of classes and relationships among its classes. It is static because the structure described is always valid at any point in the system life cycle. A system can have a number of class diagrams and a class can be seen in several class diagrams. Once class diagrams are correctly produced, the implementation becomes straightforward, provided that behaviors of each object have been well known or well defined by UML state diagrams.

It is noted that the design process starts from a use case and is iterative as well as incremental. It means steps in a process will be repeated and more details or changes can be added in each iteration.

4. Implementation

Sub-systems: There are four Java packages in ORAC OM as sub-systems. They are: (1) DramaSocket for socket communication with the DRAMA tasks; (2) Science program, GUI for starting an observation; (3) Sequence Console, GUI for controlling an observation; and (4) Helptool, providing an on-line help package.

GUI classes: All the GUI classes extend from Java Swing classes. These swing classes are JFrame, JPanel, JButton, JMenu, JTextField, JLabel, and JTree etc. Swing has advantages over Java AWT since it has better interaction with external sources e.g. a mouse, a lightweight framework with a platform specific look-and-feel and a common Java API etc.

Networking: There are two basic networking connections required by the system. One is the data transmission between the Java part of the OM and its C part of the DRAMA tasks. The second is the passing of Java objects among the different processes.

For the first, a socket connection is used. It is a server socket on the Java side, using the ServerSocket classes and a client socket on the DRAMA side. Each instrument will have one sequence console which has its own ServerSocket.

For the second, Java RMI is used because of its simplicity and Object Orientation. There are two places using RMI in the system. One is to fetch a "science program" object from Observing Tool database server. Here the OM is a client. The other is that a telescope operator runs a Java process to fetch a copy of a "sequence console" object over to his computer screen. So the OM will have to run a RMI server and the telescope operator runs a Java process as a client.

All the networking is TCP/IP based and has good reliability.

Programming Tool: Jbuilder is a visual development tool for Java programming. It support pure Java, Swing, RMI and more. It is powerful, popular, inexpensive and easy to use. We have certainly found it beneficial to use because of its IDE and its debug features.

5. Testing

Each Java package has been individually tested. Each Java package has a "driver" class to test its functionality.

The system has been tested and integrated with other systems e.g. ORAC sequencer, UKIRT Data Handling System, UKIRT Telescope Control System and an instrument control system. Real observing has been successfully undertaken by use this system on the UKIRT.

Java source codes have been reviewed and documented by use of JavaDoc.

6. Conclusion and Recommendations

Conclusions: ORAC Observing Manager has been developed to provide Graphical User Interfaces and related networking for the new UKIRT Observing Control System, using modern software technologies e.g. UML for object oriented analysis and design as well as Java Swing and RMI for implementation. This is one of a few software systems using UML and Java for observing control.

Recommendation: We found that the object-oriented approach works well for astronomical user interface development and that UML is a powerful notation to describe/model object oriented systems. UML also helps communication among the members of the development team. Java Swing is powerful for GUI development. It has all the advantages associated with Java programming. It has very rich features for GUI development and is easy to use. Java RMI is good for networking because of its simplicity and beauty of object orientation. Service or manipulation on a remote object is almost as easy as it is locally. It is worth noting that there were some problems in the use of RMI with Swing components due to uncertainty in the serialization of swing components.

Acknowledgments. We thank the JAC and ATC scientists and software engineers for their comments. Also, we would like to thank Alan Pickup of the ATC for his work on the ORAC sequencer; Maren Purves, Nick Rees and Russel Kackley of the JAC for their work on the instrument and telescope control software.

References

Booch, G., Rumbaugh, J. & Jacobson I. 1999, "The Unified Modeling Language User Guide" Addison-Wesley

Bridger, A., Economou, F., Wright, G. S., & Currie, M. J. 1998, Proc. SPIE, 3349, 184, ed. P. J. Quinn

Farrell, T. J., Shortridge, K., & Bailey, J. A. 1993, BAAS, 25, 954

Jacobson, I., Booch, G., & Rumbaugh, J., 1999, "The Unified Software Development Process" Addison-Wesley

Quatrani T. 1998, "Visual Modeling With Rational Rose and UML" Addison-Wesley

Wampler, S., Gillies, K., Puxley, P., & Walker, S. 1997 Proc. SPIE, 3112, 246, ed. H. Lewis