Planning for Reliability

Providing Information to the Observer

MSOTCS has been designed with robustness and reliability in mind from the start. It has also been designed to provide as much information about the controlling environment as possible to the observers. For example if the telescope is tracking when the primary mirror cover is closed, a warning is generated. The system will only very rarely refuse to perform an action because of the configuration of the state of the hardware, but it will issue warnings and be as helpful as it can be to the observer. For example it is possible to operate the telescope if the instrument rotator or the telescope focus are reporting an error. MSOTCS does not attempt to second guess whether or not it is sensible to operate without rotator or focus control – it assumes that the observer is capable of making that decision. However, when it comes to situations that are potentially dangerous to the telescope, such as when observing in wind or when viewing a supernova, MSOTCS will close the dithering mirror to protect the telescope. Note that in the case where the observer is physically at the telescope (rather than operating it remotely) it is possible to override this behavior (as a side-effect of the “remote observer interface” switch).

Handling Errors and Restarting Processes

Each process that is a part of MSOTCS has been designed to be highly cohesive and loosely coupled to other processes via the control database which stores status vectors. This allows for the failure of occasional components to be handled. From this it then follows the strategy that whenever a process encounters a serious error it simply logs the details and exits. If the process is restartable, the restart is automatically performed so that the process can continue. If it is not restartable, the process is only allowed to fail once – in particular, there are only two processes required for the telescope to track (these being the tcl_control process that provides the main telescope loop), as long as these two processes continue to run, then flows the strategy that was outlined above.

MSOTCS: A New Telescope Control System for the Australian National University’s 2.3m Telescope at Siding Spring Observatory

Introduction

The Mount Stromlo Observatory Telescope Control System (MSOTCS) is software written in C++ running on the real-time operating system QNX. It has been written to accommodate a remote observing software, TAROS. MSOTCS is responsible for the control and safety of the hardware which constitutes the optical support structure of the telescope and of associated plant which contains the observing environment such as ventilation, conditioning, building/electrical isolation, shutters, instruments, safety interlocks etc. Most importantly, the TCS implements control for some mechanisms which affect telescope axis motion: rotational stages, slew, focus, reticle, mirror and mirror positioning. In addition to the control functions, the MSOTCS also provides a functionality that is unique in the world of remote observation (MSOTCS was designed with an eye to future plans for the Remote Control and Monitoring Long Baseline Interferometer). It is designed to be reusable, portable and robust.

Methods and Observers

Avoiding memory leaks

In order to avoid memory leaks we ensure that every dynamically allocated object is stored in an std::ptrc, where possible. In some cases (for example objects encoded via the BNC c++ class) this is not possible. A pointer class is implemented to add the RAII approach used here. We did not do this and it is possible to have QNX memory leak reports even when there were no memory leaks in a program. We have added code generation to the BNC c++ class that allows us to generate std::ptrc objects automatically for this purpose. Sources may wish to investigate the Boost C++ library (http://www.boost.org), which provides a scoped linked list model.

Keeping the whole system running

As MSOTCS is designed to run as an embedded system it is necessary for it to start automatically when the machine is powered on or restarted. This is done automatically using the standard init script mechanism, meaning the system running all the time is at least minimal. In addition, in order to ensure this, the whole switching process is not sufficient, as it will shut down the system if any of the critical processes fail. We implemented a simple “wake up” script that only allows the switching process to terminate if there are no other processes running.

MSOTCS: A New Telescope Control System for the Australian National University’s 2.3m Telescope at Siding Spring Observatory

Introduction

The Mount Stromlo Observatory Telescope Control System (MSOTCS) is software written in C++ running on the real-time operating system QNX. It has been written to accommodate remote observing software, TAROS. MSOTCS is responsible for the control and safety of the hardware which constitutes the optical support structure of the telescope and of associated plant which contains the observing environment such as ventilation, conditioning, building/electrical isolation, shutters, instruments, safety interlocks etc. Most importantly, the TCS implements control for some mechanisms which affect telescope axis motion: rotational stages, slew, focus, reticle, mirror and mirror positioning. In addition to the control functions, the MSOTCS also provides a functionality that is unique in the world of remote observation (MSOTCS was designed with an eye to future plans for the Remote Control and Monitoring Long Baseline Interferometer). It is designed to be reusable, portable and robust.

Methods and Observers

Avoiding memory leaks

In order to avoid memory leaks we ensure that every dynamically allocated object is stored in an std::ptrc, where possible. In some cases (for example objects encoded via the BNC c++ class) this is not possible. A pointer class is implemented to add the RAII approach used here. We did not do this and it is possible to have QNX memory leak reports even when there were no memory leaks in a program. We have added code generation to the BNC c++ class that allows us to generate std::ptrc objects automatically for this purpose. Sources may wish to investigate the Boost C++ library (http://www.boost.org), which provides a scoped linked list model.

Keeping the whole system running

As MSOTCS is designed to run as an embedded system it is necessary for it to start automatically when the machine is powered on or restarted. This is done automatically using the standard init script mechanism, meaning the system running all the time is at least minimal. In addition, in order to ensure this, the whole switching process is not sufficient, as it will shut down the system if any of the critical processes fail. We implemented a simple “wake up” script that only allows the switching process to terminate if there are no other processes running.

Why QNX?

QNX (http://www.qnx.com) provides a POSIX environment that is very like traditional Unix. Our existing C++ code was originally written under Linux but the necessity to move to QNX for TAROS made this move possible. The QNX operating system itself is fast and efficient: indeed, the QNX operating system itself is

Contact

Jon Nielsen
Research School of Astronomy and Astrophysics
ANU
Cotter Rd, Weston Creek, ACT 2611, Australia
Email: jnn2@anu.edu.au
Phone: (02) 6125-0230
Web: http://www.mso.anu.edu.au/