Astronomical Data Analysis Software and Systems XVIII
ASP Conference Series, Vol. 411, © 2009
D. Bohlender, D. Durand and P. Dowler, eds.

Hardware Device Simulation Framework in the ALMA Control Subsystem

Matias Mora and Cecilia Reyes

Computer Systems Research Group, Universidad Técnica Federico Santa María, Av. España 1680, Valparaíso, Chile

Jorge Ibsen

European Southern Observatory, Alonso de Córdova 3107, Santiago, Chile

Jeff Kern, Thomas Juerges, and Allen Farris

National Radio Astronomy Observatory, 1003 Lopezville Rd., Socorro, 87801 NM, U.S.A.

Rodrigo Araya, Nicolás Troncoso, and Víctor González

Associated Universities, Inc., Av. El Golf 40, Piso 18, Santiago, Chile

Abstract. Hardware device simulation development is a fundamental task which has to be addressed when writing control software. Simulations are used to decouple the software from the hardware layer, and provide a powerful tool to ensure the correct functionality of a control system before integrating real devices. This paper presents the design of the ALMA hardware device simulation framework as part of the Control subsystem. This framework provides basic code generation, allows simulation of devices through an external process connected to a real-time FIFO (as the real hardware), and provides an alternative, direct and more flexible simulation. This has simplified development and testing as developers can now focus on the non-trivial aspects of a simulation.

1. Introduction

1.1. The Control Subsystem

All software subsystems for the ALMA Project are being developed over a common distributed framework, called ALMA Common Software, first presented by Raffi, Chiozzi, & Glendenning (2001). ACS is an object model based on the CORBA specifications and uses the Component-Container model. It provides a common framework, from the application layer down to the hardware control.

The ALMA Control Subsystem architecture enables the control of the entire antenna array. Internal antenna devices are controlled through the ALMA Monitor and Control Bus (AMB), which is a CAN bus connecting the devices with the Antenna Bus Master (ABM). Each device has an associated ACS C++ component in charge of its control. In general terms this manages all of the device's monitor (read) and control (write) points, specified by a Relative CAN

Address (RCA). All monitor and control points are defined in each device's Interface Control Document (ICD). The component's interface is defined by its IDL, allowing other (higher-level) components and clients to interact with it, independently from the implementation language.

These control components have quite a lot of things in common: the basic C++ structure for ACS and common and specific monitor and control points management. Fortunately for the developers, it is not necessary to code these common parts again each time, thanks to a code generation framework (Farris 2006) based on openArchitectureWare. The device control components are generated based on their ICD specifications, written in XML *spreadsheets*. They contain general device attributes, monitor and control points specifications (RCA, data types, value range, etc.) and archiving information for some monitor points. At build time the code generation framework automatically creates the output classes, header files and IDLs based on generic template files written in a simple markup language. This technique has resulted in a dramatic improvement in productivity, since software developers can concentrate on higher-level aspects of the device control.

1.2. Original Hardware Device Simulator

At the beginning of the present work (early 2008), the ALMA Control subsystem was using the AMB Loopback Simulator to simulate all hardware devices and test the corresponding control components. This strategy was based on a simple C++ class, instantiating a hard-coded list of simulation classes; each one of them supported the device's monitor and control points. As an external process, it could receive CAN bus messages from the device control component through a Kernel real-time FIFO, and dispatch them to the proper simulation object. The replies went back to the component by the same mechanism.

The simulation classes involved in this strategy had to be created manually as they were needed. Since the communication with these classes was done through the RT Kernel space, this approach had the limitation of not being able to simulate more than one antenna set on a single machine. In addition, all devices were aggregated by a single process, introducing some limitations in its flexibility, losing the distributed nature of the system and making it hard to control the lifecycle of the simulation classes.

2. ALMA Hardware Device Simulator

The main goal of this new framework is to simplify the development of hardware simulations. The Control code generation framework was adapted to build the basic C++ simulation classes from the existing spreadsheets. These classes could be further extended by developers to add special functionality. The new simulation classes are backwards compatible and can be run either with the traditional Loopback Simulator or a direct simulation, using an alternative communication class which replaces the hardware layer.

2.1. Design Overview

A general class overview of the new simulation strategy is shown in Figure 1. Black classes are part of the hardware control module, green ones are the

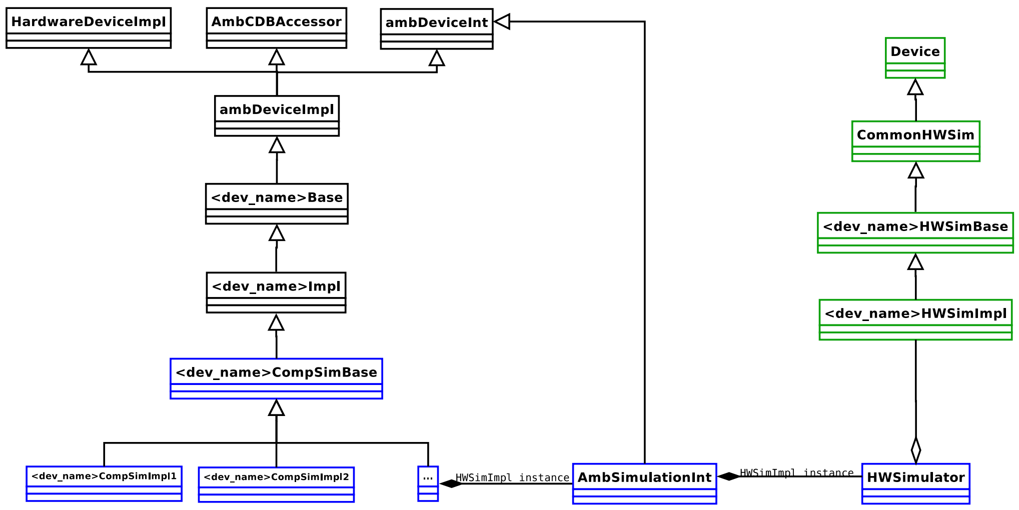

Figure 1.: Main class diagram for the ALMA hardware simulation system.

simulation classes and blue ones are part of the simulation component extension of the actual hardware control component.

The hardware control component consists of a generated `<dev_name>Base` class and a user-extended `<dev_name>Impl`. In the new simulation strategy, a `<dev_name>CompSimBase` class extends the original control component, changing only the `ambDeviceInt` communication methods by the ones provided by `AmbSimulationInt` which is used through composition instead of inheritance. Both implement the same methods, although `AmbSimulationInt` interacts with the `HWSimulator` instead of the RT-FIFO. Also, two new methods are added to allow simulating error conditions at runtime.

On the simulation side, common monitor and control points are managed in the `CommonHWSim` class, while `<dev_name>HWSimBase` is generated for each device, providing specific points simulation. Simulation logic is further introduced by developers in the `HWSimImpl` class. A sequence diagram of the simulation process is shown in Figure 2.

The `<dev_name>CompSimBase` and `<dev_name>CompSimImpl` classes are also generated for each device. The last one contains only the constructor with the instantiation of the `<dev_name>HWSimImpl` class. The separation was done to allow the developer to provide alternative simulations by instantiating different `HWSimImpl`-like objects with different behaviors.

3. Conclusions

The new ALMA hardware simulation strategy enables a device control component to simulate itself, without passing through the RTOS channel and without any external running process. Also, mainly through the usage of code generation, a simulation framework is provided which enables the developers to focus only on adding simulation functionality. This has once again reduced the development time for a control component dramatically, as a default simulation is provided *out of the box*.

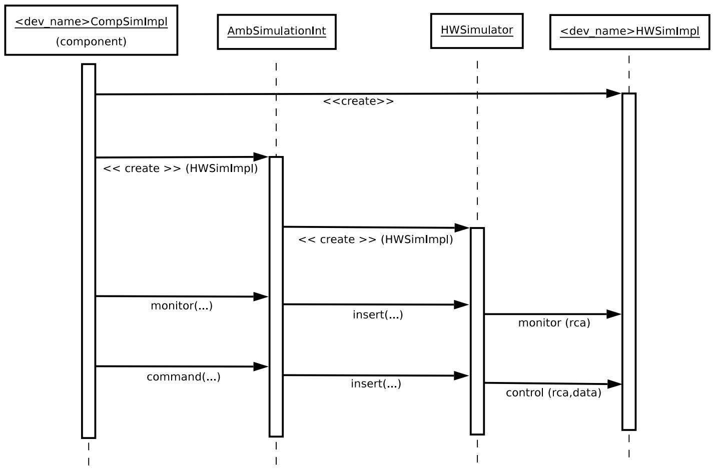

Figure 2.: Sequence diagram of the ALMA hardware simulation process, without RT-FIFO.

The new framework is currently in use for all hardware devices of the Control subsystem which use the CAN bus. The non-RTOS simulation has so far allowed mounts of simulated arrays of up to four antennas on a single machine. Also, nearly all development and debugging of control components, before plugging them into real hardware, is now done on personal workstations without RT requirements, instead of specialized servers. A review of current simulation and testing levels in the Control subsystem is presented in Hiriart & Kern (2009).

Finally, the proposed solution has some design problems, mainly because it had to be integrated into the existing Control subsystem infrastructure, with as few changes as possible for the developer, so as not to cause problems in current critical development phases. The Computer Systems Research Group at UTFSM is working on a generalization of this simulation framework to form part of a telescope simulation environment to be attached to a generic telescope control model (Tobar et al. 2008).

Acknowledgments. This work was supported by ALMA-CONICYT Fund project #31060008 “Software Development for ALMA: Building Up Expertise to Meet ALMA Software Requirements within a Chilean University”, under development at Universidad Técnica Federico Santa María, and NRAO, through Associated Universities, Inc.

References

Farris, A. 2006, in ASP Conf. Ser. 376, ADASS XVI, ed. R. A. Shaw, F. Hill, & D. J. Bell (San Francisco: ASP), 523

Hiriart, R. & Kern, J. 2009,in ASP Conf. Ser. 411, ADASS XVIII, ed. D.A. Bohlender, D. Durand & P. Dowler (San Francisco: ASP), 414

Raffi, G., Chiozzi, G., & Glendenning, B. 2001, in ASP Conf. Ser. 281, ADASS XI, ed. D. A. Bohlender, D. Durand, & T. H. Handley (San Francisco: ASP), 103

Tobar, R., von Brand, H. H., Araya, M. & Lopez, J. 2008, in Proceedings of SPIE 2008, Advanced Software and Control for Astronomy, 7019