DESIGN STRATEGIES IN THE DEVELOPMENT OF THE ITALIAN SINGLE-DISH CONTROL SYSTEM

A. Orlati, M. Bartolini, S. Righini, INAF-IRA, Bologna, Italy
M. Buttu, A. Fara, C. Migoni, S. Poppi, INAF-OAC, Cagliari, Italy

Abstract

The Italian National Institute for Astrophysics (INAF) manages three radio telescopes: the Medicina and Noto dishes and the newly-built SRT. In order to make their capabilities more valuable to the scientific community, we started the DISCOS (Development of the Italian Single-dish COntrol System) project. DISCOS is implemented according to a distributed Component-Container model and hides to the users the differences among the telescopes by presenting the same user interface and the same data format. The complexity of coping with three heterogeneous instruments was handled designing a software development infrastructure with a wide monolithic codebase (libraries, components and generic interfaces), which is completely shared among the three product lines. This design permits to produce new software components with a minimum effort and to set up the same test suites for all the environments, thus leading to an affordable development and maintenance process. In this paper we illustrate the design strategies and the development techniques used to realize and optimize this common control software. We also provide a description of the project status and future plans.

INTRODUCTION

The National Institute for Astrophysics (INAF) manages three radio telescopes in Italy: The Sardinia Radio Telescope (SRT) [1] and the Medicina and Noto 32-m dishes. The newly-built SRT, located in the Sardinia island, was inaugurated in 2013. Since the early stages of the project, staff from all the three telescopes has been involved in the development of the SRT control software. This forced us to cope with both the development of a brand-new system and the maintenance of the already existing telescopes, compelling the involved personnel to learn and enhance their competence in distinct and heterogeneous systems based on completely different technologies. Developing a control software to be installed at all the telescopes was the natural and straightforward approach to this problem.

This idea was formalized with the creation of the Development of Italian Single-dish COntrol System (DISCOS). The project aims to provide all three Italian radiotelescopes with a common infrastructure that increases their capabilities and the usability for the scientific community and, at the same time, optimizing the efforts made by the technical staff for its development and maintenance.

DISCOS is based on ALMA Common Software (ACS) developed at ESO for the ALMA project [2]. ACS implements a Component-Container model via CORBA (Common Object Request Broker Architecture). It provides a set of tools, libraries and development patterns that hide the complexity of CORBA. This approach permits to reduce the time required for coding and development. The framework also supports both real and non-real time platforms for C++, Python and Java.

The core of the project was completed during the main development stage. The SRT and Medicina installations are now fully operational and supervise all the telescope operations. The Noto antenna is also equipped with a preliminary version, which is going to be finalized in a few months.

In this paper we describe the salient design choices that allowed us to reuse the most part of the code for all the production lines and to hide the instrument complexity under a common infrastructure. We also illustrate the maintenance and development workflows, as they are fundamental part of the DISCOS design.

THE TELESCOPES

The INAF radiotelescopes share some common aspects but are in general different from one another. Medicina and Noto have passed through years of development and updates, while SRT is the result of an organic employment of state-of-the-art technologies. Table 1 recaps the main characteristics and the differences, listing all the major sections – servo systems, front-ends, back-ends. All the telescopes are presently participating in the VLBI network. Medicina and Noto are also offered to the scientific community as single-dish facilities (SRT will very soon be, as well).

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

Table 1: Fact Sheet of the Three Italian Radiotelescopes

	SRT	Medicina	Noto
Main mirror	shaped profile, 64 m	parabolic profile, 32 m	parabolic profile, 32 m
Optical configuration	Gregorian	Cassegrain	Cassegrain
Mount	Altazimuth, fully steerable 12 motors + cable wrap	Altazimuth, fully steerable 4 motors	Altazimuth, fully steerable 4 motors
Antenna Control Unit (main servo system)	Beckhoff PLC ethernet vendor protocol	VxWorks based PC ethernet vendor protocol	VxWorks based PC ethernet vendor protocol
Primary Focus	three degrees of freedom INAF protocol	three degrees of freedom INAF protocol	three degrees of freedom INAF protocol
Secondary Focus	six degrees of freedom ethernet INAF protocol	five degrees of freedom ethernet INAF protocol	five degrees of freedom RS232 vendor protocol
Active Surface	1008 aluminium panels 1116 actuators rs485/ethernet vendor protocol	Not available	240 aluminium panels 244 actuators rs232 vendor protocol
Receivers (RF bands*)	(0.305-0.410) (1.3-1.8) (5.7-7.7) (18.0-26.5), 7 feeds GPIB and ethernet INAF protocol	(1.35-1.45) (1.595-1.715) (2.2-2.36) (4.30-5.80) (5.90-7.10) (8.18-8.98) (18.0-26.5), 2 feeds GPIB, ethernet and RS232 various protocols	(0.317-0.320) (1.40-1.72) (2.20-2.36) (4.70-5.05) (8.18-8.58) (22.18-22.46) (39.0-43.3) GPIB and RS232 various protocols
Backends (Bandwidth*)	TotalPower [continuum] (up to 2.0), 1-1000 ms, 14 inputs XARCOS [spectro-polarimetry] (up to 0.125), 10 s, 2048 bins, 14 inputs Roach [spectro-polarimetry] (0.512), 10-1000 ms, 8192 bins, up to 14 inputs DFB [spectro-polarimetry] (1.024), 1-4000 ms, 8192 bins, 4 inputs	TotalPower [continuum] (up to 2.0), 1-1000 ms, 4 inputs XARCOS [spectro-polarimetry] (up to 0.125), 10 s, 2048 bins, 14 inputs	TotalPower [continuum] (up to 2.0), 1 ms, 4 inputs

* Frequencies are expressed in GHz.

A COMMON STRATEGY

Broad categories like development, operations and maintenance are very demanding, especially when highly qualified expertise is required. Our common infrastructure enhances the quality and throughput of our work.

Operations

Operations are immediately affected by this strategy; as operators can now manoeuvre the three telescopes facing very slight differences in the user interface, their require a single training. Similarly, astronomers are presented with the same observing modes and the same data format at every telescope. Operation manuals and documentation are written with a common effort, sharing most of the information among the infrastructures.

Maintenance

Maintenance consists in code debugging, configuration changes or even complete replacements of software components. Since regular telescope activities require to minimize the time devoted to maintenance, the relative work must be carried out with proper planning and efficiency. This is eased by the use of the same ticketing

ISBN 978-3-95450-148-9

Copyright © 2015 CC-BY-3.0 and by the respective authors

system to track software bugs and to monitor the antenna status. A common workflow, involving every collaborator, is employed to fix malfunctions. As the needed tests can be performed at any telescope, we can choose the one providing an adequate scheduling to test patch releases.

Development

The development of new software features is a time-consuming activity that also requires testing time. The DISCOS common platform highly reduces these needs. New components can be deployed indifferently at all the antenna systems, so that separate units can independently carry out the development work. Moreover, once a new piece of hardware is commissioned and integrated into the software system of one of the telescopes, it can be replicated at the other telescopes with little effort.

DESIGN

The design of the DISCOS control software highly relies on the ACS patterns and services. In the ACS model, the basic unit performing a task is a component. It either controls a simple device or executes astronomical computations. Each component exposes an interface or list of capabilities and is individually configured to determine its exact behaviour inside the system. In our project the components are organized into subsystems or packages according to their functional affinity, thus forming system segments that are independent and capable to run as stand-alone groups. During the early development efforts this configuration turned out to be very valuable for our geographically-spread team. Each developer, in fact, could focus on one single subsystem, internal milestone or scheduled test, without being concerned by the delays in the development of the other packages.

Supporting three different telescopes is of course the most challenging goal of the project. This can be reached by expanding the common code, the part of our project that can be reused and deployed at all sites, as much as possible. The station-specific modules, then, consist essentially in the low-level and no-logic control of the devices and of the telescope hardware. According to this approach, a careful design was crucial to ensure that all the peculiarities of the telescopes fitted with the business logic at a higher level and with the common parts of the control software. The Italian telescopes differ in various features (see Table 1), which can be grouped into two categories:

- A given telescope functionality is implemented via different hardware devices at the three radiotelescopes: they could differ in vendor, working mode and communication protocol. The main servo systems of our antennas are an excellent example of this case; in our software implementation such devices were modelled according to an ideal or generic interface that defines the minimal set of attributes and methods required by the control logic. Every other component handles the device adopting this interface.
- An apparatus is installed at one or two telescopes, but it does not apply to other antennas. For example, the SRT and Noto antennas are equipped with an active surface, while the Medicina one is not. These categories are handled through configuration files describing the available components and functionalities; the system is then capable of detecting if an operation is allowed under the present configuration. If the operation is not fundamental for the running observation, no error is raised.

Development Workflow

In recent years we have tried to formalize the development workflow of new components, resulting in clearer and more maintainable code. As a first step we decided to add the ability to run components by simulating the hardware layer whenever possible. This led us to split the component development into:

- Development of a hardware simulation server
- Development of a standalone hardware communication library
- Development of the ACS component exploiting the library

The immediate benefit of this procedure is to have a simulated hardware environment that permits to run our tests automatically. We consequently adopted integration tests to validate component interfaces and regression tests when fixing bugs in the system. Furthermore, we set up an infrastructure - both Python and C++ are supported - that generates a testing skeleton for new components and runs unit tests collecting xUnit-formatted results.

A new approach was adopted also in the configuration of the software environment. The full stack deployment was formalized via code, resulting into the AZDORA project [3]. It uses virtualization and provisioning technologies in order to setup and run a fully configured DISCOS environment, which can be used both for development and production workstations. This uniformity among development, testing and production environments is crucial in assessing the software stability as much as possible before the on-field production installation.

The just-described features have made it possible to adopt Continuous Integration practices in the development process. A Jenkins server [4] is used in this case. At present we run nightly builds of the trunk branch of each telescope and of every still-maintained release. The proper developer is alerted in case problems are detected; occasional build or integration errors are fed into our bug-tracking system. Each build target results into a ready-to-install package that can be safely used in production environment. This infrastructure also permits test automation and to collect the results on the whole project. In the near future this will give access to new quality metrics and to the possibility of automatically

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

running acceptance tests upon every new code subscription.

Maintenance Workflow

System bugs are handled with regression tests: each time a bug is found, the developer writes a test that reproduces the buggy behaviour, then the fix is not committed into the patch release until the test executes correctly.

We also adopt a *major.minor.patch* nomenclature. The major version zero (0.y.z) is for initial development, when anything may change at any time: the public APIs, the operator input commands and the schedule grammar are not to be considered stable. New major versions are issued when non-backward-compatible changes are necessary and are planned way in advance. Minor versions are feature releases, so they add functionality in a backward-compatible manner. We also have non-production ready versions that get an additional qualifier: beta and release candidate (RC). The beta versions are to be tested by advanced users, while the RCs are aimed at being tested by a group of astronomers.

Release notes are written by developers in a user-friendly manner and can be found within our documentation portal [5], while technical release notes are automatically extracted from the bug-tracking and features system used to manage the development process [6].

FUTURE PLANS

The DISCOS user interface is currently mainly composed of unrelated GUI applications showing textual information about the main telescope components via *ncurses* based panels. Every interaction with the control software is performed via a command shell that serially reads telescope commands in a synchronous *Read-Evaluate-Print* loop. A remote access to the interface is offered via VNC connections. This setup is effective for early operations; in view of the telescopes future activities we are designing a user-friendly interface in the form of a web application. This will provide the users with a more homogeneous experience, efficiently enabling remote operations. This approach will likely require other software layers, implementing the authentication and authorization of the users and new parameters-publishing systems. This will also lead to a possible integration with other telescope management software tools, such as the proposal submissions handling system and the schedule creator. This work could also technologically enable the creation of a centralized operation centre for all the radiotelescopes.

Further development in the near future will involve the adoption of the most recent ACS version, the adoption of new technologies for high-rate data transfer, the extension of test coverage to a bigger portion of the code and the automation of tests during the build process.

CONCLUSIONS

DISCOS is serving the scientific community by enabling observations at the Italian INAF radiotelescopes, and while being still subject to some major changes, it is running and effective. The growth of the project and its dissemination among different sites has forced the adoption of a common development strategy and has led to the adoption of best practices from the software industry such as Test Driven Development and Continuous Integration in order to assess software quality and to ensure the stability of successive software releases. This also enables an effective maintenance process, which is a strong assurance for the coming years of development and operations.

REFERENCES

[1] G. Grueff et al, “Sardinia Radio Telescope: the new italian project”, Proc. SPIE vol 5489, p 773-783 (2004)

[2] G. Chiozzi et al., “The ALMA common software: a developer-friendly CORBA-based framework”, Proc. SPIE vol 6274, September 2004, p. 205 (2004)

[3] Azdora project github page: https://github.com/discos/azdora

[4] Jenkins project website: https://jenkins-ci.org/

[5] DISCOS documentation portal: http://discos.readthedocs.org/en/latest/developer/releasenotes/releasenotes.html

[6] DISCOS project bug tracking system: http://www.med.ira.inaf.it/mantisbt/roadmap_page.php

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9