Diseño e implementación de un observatorio robótico teleoperado basado en Robot Operating System

Design and implementation of a robotic observatory teleoperation based on Robot Operating System

Andrés Villalobos Rivera¹ José Gallardo Arancibia^{2*}

Recibido 27 de junio de 2018, aceptado 06 de agosto de 2018

Received: June 27, 2018 Accepted: August 06, 2018

RESUMEN

Un observatorio astronómico requiere de cielos nítidos y nula contaminación lumínica para cumplir eficazmente su cometido, razones por las que debe ser ubicado en zonas alejadas de áreas urbanas y/o áreas pobladas y, por tanto, frecuentemente deben ser operados remotamente. En este trabajo, se postula la hipótesis de que el framework de Computación Distribuida “Robot Operating System (ROS)”, utilizado en el desarrollo de la robótica, permitiría la construcción de sistemas de teleoperación, si se homologa la estructura de un telescopio con la estructura de un Robot. La hipótesis se verifica, desarrollándose los drivers necesarios, para operar remotamente el observatorio ubicado a 90 kilómetros de la ciudad de Antofagasta. Los resultados obtenidos, permitieron confirmar la hipótesis, lográndose teleoperar el observatorio “Ckoirama” de la Universidad de Antofagasta (ubicado a 90km de la ciudad de Antofagasta), desde una sala de control situada en la ciudad a través de internet y una red privada virtual (VPN) vía señal 3G. Los exitosos resultados permiten potenciar el desarrollo de sistemas de control para observatorios remotos, sin necesariamente depender del escaso y costoso personal experto en astroingeniería, pues existiría mucho desarrollo reutilizable en ROS.

Palabras clave: ROS, astronomía, observatorios, teleoperación, TCS, Linux, Raspberry PI.

ABSTRACT

An astronomic observatory requires clear skies and null photopollution to fulfill efficiently its purpose, reason why these buildings must be located isolated and far away from urban areas, hence, frequently remotely operated. In this paper, we propose as a hypothesis the use of a distributed computing framework called “Robot Operating System (ROS)”, widely used on robotic research, this framework would allow the development of teleoperation system by homologating the telescope structure with a robot. This hypothesis was verified by the development of all the required drivers to remotely operate an observatory located 90km away of Antofagasta. The outcome allowed us to confirm the hypothesis being able to remotely operate “Ckoirama” observatory from Universidad de Antofagasta (observatory located 90km away of the city), the observatory is being operated on the Control Room in Antofagasta and the connectivity is provided by 3G over a virtual private network (VPN). This successful outcome would potentiate the development of control systems for remote observatories, without the restriction of the small and expensive Astroengineering staff because there are several developments on ROS which can be reused.

Keywords: ROS, astronomy, observatories, teleoperation, TCS, Linux, Raspberry PI.

¹ Universidad Católica del Norte, Facultad de Ingeniería y Ciencias Geológicas, Departamento de Ingeniería de Sistemas y Computación, Avenida Angamos 0610, Antofagasta, Chile, 1240000. E-mail: avillalobos@ucn.cl

² Universidad Católica del Norte, Facultad de Ingeniería y Ciencias Geológicas, Departamento de Ingeniería de Sistemas y Computación, Avenida Angamos 0610, Antofagasta, Chile, 1240000. E-mail: jgallardo@ucn.cl

* Autor de correspondencia

INTRODUCCIÓN

En astronomía, existen dos grandes tipos de observatorios: radiotelescopios y ópticos, ambos son afectados por la densa capa de atmósfera que se encuentra a nivel del mar. Respecto de factores de contaminación, el primero es afectado por ondas de radio emitidas por los aparatos electrónicos, mientras que el segundo por la luminosidad; razones por las cuales, observatorios profesionales deben ser instalados distantes de centros urbanos, generalmente en altas montañas.

Un observatorio, consta de variables que deben ser controladas al igual que en un sistema robótico. Lograr el soporte requerido para un software de control en astronomía es muy costoso, principalmente porque según explica Gianluca Chiozzi en [1], la comunidad Astronómica es reducida, distribuida y altamente especializada, por tanto es vital el idear soluciones que permitan enfrentar este problema a un menor coste, sobretodo en observatorios pequeños y escasos de recursos financieros. ROS es un framework de software libre, soportado por una amplia comunidad de desarrolladores en el mundo entero.

En este trabajo se propone utilizar “Robot Operating System” (ROS)[2] para desarrollar sistemas de control para observatorios. Un enfoque similar es presentado en [3], en el cual un pequeño observatorio es automatizado utilizando ROS. El observatorio en el cual se validó la hipótesis fue el observatorio “Ckoirama”, de la Universidad de Antofagasta, ubicado a 90km de la ciudad. Este observatorio cuenta con un telescopio principal de 24”, un telescopio secundario de 10” y un domo corredizo tipo roll-off diseñado en la universidad. La arquitectura de software, considera la creación de aplicaciones “Clientes” que son los que utilizaría un operador y aplicaciones “Servidores” que son los “ROS Drivers”, software que se ejecuta en un computador Raspberry PI llamado Local Control Unit (LCU) y que permite a un operador a través de aplicaciones clientes, controlar un hardware en particular.

El presente artículo se organiza de la siguiente manera, en la siguiente sección se describe qué es y cómo funciona Robot Operating System (ROS). El capítulo siguiente entrega una breve descripción del observatorio utilizado como caso de estudio y

la arquitectura propuesta. Luego, se entrega una comparación con otros sistemas de control vigentes para observatorios seguido de una discusión de resultados para finalizar el texto con las conclusiones del artículo.

ROBOT OPERATING SYSTEM

Robot Operating System (ROS) [2] es un proyecto de código abierto y un meta-sistema operativo para robots. ROS provee los servicios que típicamente provee un sistema operativo incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comúnmente utilizadas, traspaso de mensajes entre procesos y administración de paquetes. Además, provee herramientas y librerías para obtener, construir, escribir o ejecutar código entre múltiples computadores, por lo tanto, se considera un Sistema de Computación Distribuida. Su principal objetivo es fomentar y soportar el reúso de código en la investigación y desarrollo de la robótica. Al ser ROS un framework de procesos distribuidos, permite que programas ejecutables puedan ser diseñados individualmente y acoplados débilmente en tiempo de ejecución. En la Figura 1 se ilustra el funcionamiento de ROS y las formas de comunicación entre nodos. Dichos procesos pueden ser agrupados en paquetes, los cuales son fácilmente compartidos o distribuidos. Para una mayor comprensión de ROS y todas sus bondades, es posible consultar [2], [4].

Figura 1. Diagrama de funcionamiento de ROS Observatorio TELEOPERADO.

El observatorio “Ckoirama” de la Universidad de Antofagasta, se encuentra ubicado a 90km de Antofagasta, en la antigua estación Yungay y cuenta con:

- Un telescopio principal de 24”,
- Un telescopio secundario de 10”

- Un domo corredizo tipo roll-off diseñado en la Universidad.

El domo está dotado de un motor Nice RUN 1500, mientras que la puerta lateral (utilizada para dotar al observatorio de un mayor ángulo de observación al sur), está dotada de un motor Nice ROBUS 600. Ambos motores poseen una tarjeta controladora que puede ser intervenida electrónicamente para conectarla a otros dispositivos.

La conectividad del observatorio es provista a través de una señal 3G, utilizando un router industrial Robustel 3PWR3000 y un plan de datos con una determinada cuota de navegación. Este router es capaz de proveer conectividad vía cable y Wifi. Para medir las condiciones atmosféricas se utilizó la estación meteorológica Davis Vantage Pro 2, esta es leída a través del software Weewx. Se utilizaron también sistemas de procesamiento tipo Raspberry PI modelo B, como nexo entre el hardware y el software.

El observatorio robotizado utilizando ROS, está pensado para funcionar dentro de una red local, y asume que puede ver cualquier puerto dentro de un rango definido en ROS. Esto significa que

no debería haber una restricción de puertos entre los nodos que están dentro del sistema, lo que no sucede en equipos distribuidos geográficamente. Para resolver este problema y lograr utilizar ROS como framework de teleoperación se utilizó una VPN. La VPN crea un túnel virtual entre los nodos conectados agregándolos a una red local virtual. En la Figura 2 se explica gráficamente el funcionamiento del sistema usando ROS y la VPN.

Respecto de la montura del telescopio, la comunicación es a través de una conexión serial con la montura Celestron. Se desarrolló un driver en C++ utilizando la librería serial provista por ROS para establecer la comunicación con el dispositivo mediante el protocolo del fabricante. Cabe destacar que el hecho de que existiese una librería serial en ROS, facilitó de gran medida el desarrollo de los drivers.

Para tener la medición del clima, se utilizó la estación meteorológica Davis Vantage Pro 2, la cual posee una serie de sensores, entre ellos medición de radiación UV, velocidad del viento, humedad y temperatura por citar algunos. Para integrar esta estación meteorológica al sistema, se utilizó un software llamado Weewx, el que se comunica periódicamente con la estación

Figura 2. ROS a través de VPN.

meteorológica, obtiene los datos y se genera una página web con los resultados, pero además se genera una base de datos con toda la información meteorológica. Por lo tanto, el ROS driver desarrollado se comunica con la base de datos generada por Weewx para obtener los datos.

El sistema fue construido en paquetes que luego pueden ser utilizados de forma independiente, es decir, no necesariamente un nuevo observatorio utilizará completamente el sistema. Pero tal vez, sí alguno de sus módulos de forma independiente, es por esta razón que se desarrollaron 3 módulos principales, aquel que controla el domo, el telescopio y los sensores llamados **dome**, **telescope** y **sensors** respectivamente. De esta forma, en algún otro proyecto se podría requerir de la operación de un telescopio sin necesariamente contar con un domo o una serie de sensores, esta flexibilidad también permite agregar nuevos drivers, lo que implica una mayor compatibilidad del sistema, siguiendo la lógica de ROS de ser un repositorio de software para un propósito determinado.

Figura 3. Descomposición en capas para un driver ROS.

Los ROS drivers, son aplicaciones que consideran la interacción directa con el hardware a ser controlado, como se muestra en la Figura 3. Lo más importante es el tipo de hardware y la forma en la que este dispositivo puede ser conectado con algún nodo computacional, esto va a definir el tipo de hardware a ser utilizado como LCU o ROS driver. En nuestro caso de estudio, todas las conexiones fueron del tipo USB, por lo tanto, una Raspberry PI es suficiente tanto en poder de cómputo como en tipo de conexiones.

Una vez determinada la forma en que estos equipos pueden interactuar a nivel físico, se construyeron las interfaces de software para entender el protocolo de hardware, por ejemplo, la montura LosMandy no utiliza el mismo protocolo que usa Celestron a la hora de comunicarse con un dispositivo. Sin embargo, comparten el mismo tipo de conexión física, USB. Luego de conseguir comunicación y entendimiento de los mensajes intercambiados entre la Raspberry PI y el hardware, se debe integrar este software a ROS para que estos mensajes, leídos desde el hardware e interpretados por Raspberry PI puedan ser entendidos por la infraestructura de ROS. La Figura 3 muestra de forma extendida como un ROS Driver, descrito en la Figura 4 es implementado.

Figura 4. Sistema de control basado en ROS.

A modo de ejemplo, en la Figura 5 hay un diagrama de como se implementó el control del domo y la puerta lateral. El Telescope Control System (TCS) consiste en un script en Python que puede ser ejecutado en modo gráfico o en modo comandos, para ambas modalidades posee las mismas capacidades en cuanto a envío de comandos y telemetría. El TCS envía comandos a través de “servicios” provistos por los diferentes ROS Drivers y lee el estado de los equipos y sensores a través de tópicos en los cuales cada ROS driver es un publicador. Los ROS drivers de la puerta lateral y el domo implementan ambos el servicio de abrir y cerrar. Además, hay un servicio de emergencia para detener cualquiera de las acciones en ejecución. Una vez el TCS invoca a alguno de los servicios, la petición llega primero al ROS Master quien es el encargado de enviar dicha petición al nodo correspondiente. Por otra parte, el ROS driver ejecuta la acción programada

y vinculada con el servicio invocado, en ambos casos, la acción se traduce en accionar un relay conectado directamente a la botonera de ambos motores, la acción del servicio termina cuando el sensor final de carrera es activado, indicando que la puerta está abierta o cerrada. Una vez finalizada la acción realizada por el servicio, esta es informada al cliente que lo invocó. Toda esta comunicación sucede en primera instancia a través de TCP/IP utilizando ROSTCP, el medio físico es usualmente Ethernet, aunque también se utiliza WiFi para la LCU del domo. Cabe destacar que los servicios pueden residir en el mismo nodo computacional o en uno remoto, para efectos prácticos, se asume que el servicio existe dentro del ambiente de ROS independiente de su ubicación física. De igual manera, cada uno de los bloques descritos en la Figura 5 podrían o no estar en la misma máquina, en el mismo lugar físico e incluso podrían encontrarse geográficamente distribuidos, mientras estos nodos existan dentro del ambiente de ROS pueden ser utilizados por cualquiera de los clientes que lo requieran, este efecto se logra usando la VPN descrita en la Figura 2.

Figura 5. Ejemplo de implementación para el control del observatorio.

COMPARACIÓN CON OTROS SISTEMAS

En comparación con otros sistemas estudiados, ROS tiene como ventaja el costo de desarrollo y mantención. Mientras que para otras plataformas propietarias como TheSkyX o ACP, hay una variedad de hardware ya integrado y funcional, dichas plataformas no están pensadas para montar un observatorio completo, más bien se dedican a compatibilizar muy bien una lista de componentes individualmente. Por otra

parte, hay un par de sistemas que sí están pensados para observatorios como ASCOM, RTS2 e INDI Library, pero son muy poco conocidos fuera del ámbito astronómico, lo que implica que el soporte de software no es tan amplio como el que posee ROS. En Astronomía el soporte es un gran problema, puesto que hay muy pocas personas que se especializan en un tipo de software particular cuyo campo está limitado solo a la astronomía.

Para cuantificar y demostrar el concepto de reutilización de código, se cuantificaron: 17 archivos Python, 7 archivos para C++ (entre archivos .cpp y .h), 15 archivos de configuración (definición de mensajes y servicios). Estas cifras se encuentran condensadas en la Tabla 1.

Tabla 1. Comparación cantidad de líneas de código.

Tipo	Líneas de Código		Total
	ROS	Ckoirama	
Archivos de Python	79.935	1.450	81.385
Archivos de C++	73.818	1.512	75.330
Archivos de Lisp	8.125	0	8.125
Archivos de Configuración	1.980	54	2.034
Total	164.858	3.016	166.874

El código desarrollado para el sistema de control del observatorio es de 3.016 líneas de código, entre archivos de Python, C++ y archivos de configuración. ROS por su parte, considerando el framework base y los utilitarios instalados disponibles en los repositorios, posee 344 archivos Python sumando un total de 79.935 líneas de código, 379 archivos C++ sumando un total de 73.818 líneas de código y 203 archivos de configuración sumando 1.980 líneas de código y comentarios. A estas estadísticas no se agregaron los archivos .xml, .yaml, Makefiles, cmake y cualquier otro archivo de configuración que no sea parte de la configuración para construir el sistema. En la Tabla 2 se consigna la cuantificación del porcentaje del sistema que debe ser mantenido en el observatorio Ckoirama.

La arquitectura del software desarrollado, es de tipo Cliente-Servidor basados en la arquitectura

Tabla 2. Comparación de líneas de código en porcentajes.

Tipo	Porcentajes	
	ROS	Ckoirama
Archivos de Python	98,21%	1,78%
Archivos de C++	97,99%	2,00%
Archivos de Lisp	100%	0%
Archivos de Configuración	97,34%	2,65%
Total	98,19%	1,80%

de computación distribuida inherente a ROS como se expuso en la sección de Robot Operating System.

En la Figura 1, se consideró un nivel dentro del software que está encargado única y exclusivamente de la interacción con el operador o usuario, los llamados “*ROS Clients*”, en este nivel las instrucciones son basadas en objetivos que el científico o el operador pretende lograr, cosas como las coordenadas del objeto que desea observar, el tiempo de tracking, el tiempo de exposición, el tipo de filtro de la cámara, por dar algunos ejemplos. Todas estas instrucciones son luego traducidas a operaciones del observatorio, donde cada LCU o ROS Driver está encargado de revisar y/o controlar algún dispositivo, y es así que:

- La LCU de la montura es responsable de consultar el estado de la montura con la frecuencia requerida, en qué formato enviar los comandos a la montura y cualquier otro control o medición que esta requiera.
- La LCU de la cámara se encarga de establecer comunicación con la cámara y enviar usando un protocolo definido, las instrucciones correspondientes como el tiempo de exposición o el filtro a ser utilizado, una vez la imagen es obtenida, esta se transmite al servidor que corresponda.
- La LCU de la estación meteorológica debe ser capaz de proveer la información relacionada con el clima y alertar al resto del sistema cuando la situación se torna insegura.

DISCUSIÓN DE RESULTADOS

Se realizó un exhaustivo estudio sobre diferentes sistemas utilizados en astronomía, su funcionamiento y características, uno de los factores comunes

encontrados en la mayoría de los sistemas, es que funcionan como sistemas de computación distribuida, algunos ejemplos que pueden mencionarse se presentan en [5]-[12], lo que implica que se tiene una gran cantidad de equipos computacionales trabajando en conjunto. Otro factor común encontrado, es el uso de Linux como sistema operativo base, lo que otorga una mayor flexibilidad en el diseño y desarrollo de aplicaciones. Todos los sistemas analizados utilizan aplicaciones clientes y servidoras, las aplicaciones clientes tienen como principal función la interacción del operador con el sistema en un formato que el operador o científico pueda entender. Por otra parte, las aplicaciones que son servidoras, se encargan de transformar esos requerimientos en una serie de comandos que pueden ir directamente al hardware o bien pasar por uno o varios otros servidores. Con esta información se realizó una comparativa donde se determinó la factibilidad de utilizar ROS en la teleoperación de observatorios.

Un observatorio astronómico es un sistema que opera elementos grandes y pesados con altísima precisión como se menciona en [5], por lo tanto, para cumplir de la mejor forma posible este requerimiento, es necesario que las operaciones de control sean locales, mientras que las consignas sean calculadas y enviadas a las LCU desde un servidor con mayor poder de cómputo, tratando de evitar al máximo la comunicación remota y cálculos innecesarios en las LCU para las operaciones de control de precisión. Otro factor importante a considerar para realizar la teleoperación usando ROS, es que ROS está diseñado y desarrollado para ser utilizado en una red local, por lo tanto, para resolver este problema, todos los nodos que interactúan entre si deben estar conectados a la misma VPN para generar una red local virtual utilizando como medio la VPN, de esta forma se protege la transmisión de información sensible y se mantiene el concepto y diseño de ROS habilitándolo para locaciones remotas.

Utilizando el concepto de reutilización de software, se puede afirmar de acuerdo a lo consignado en la Tabla 2, que la Universidad de Antofagasta para su observatorio Ckoirama, debe hacerse cargo de un 1,80% del total del sistema, esto significa que el otro 98,2% está provisto y ya soportado por ROS y su comunidad. Este es un factor clave, en otros observatorios el problema es siempre el mismo, el software de control está desarrollado especialmente

para un determinado observatorio y sus propósitos particulares, lo que implica que el soporte debe ser provisto por los mismos profesionales que trabajan en el observatorio, lo que dificulta el uso de estos sistemas en otros observatorios por la dificultad de implementación y desarrollo. Sin embargo en el sistema desarrollado para la Universidad, no se depende de ningún desarrollador en particular, un desarrollador con conocimientos en ROS sería capaz de tomar control del soporte, desarrollo y mejora del sistema de control sin antes haber tenido experiencia previa en Astronomía, esto gracias a que ROS es un framework de robótica y no de astronomía propiamente tal.

Como evidencia de esta última aseveración, durante el desarrollo de este proyecto, se recibieron varias actualizaciones de ROS las cuales fueron instaladas en el sistema de control, esto significa que gente trabajando en proyectos de robótica, implícitamente brindaron soporte y actualizaciones a un observatorio por el solo hecho de actualizar y trabajar en el framework.

Al momento de iniciar este proyecto, la versión disponible de ROS era Groovy, pocos meses después se hizo el release oficial de ROS Hydro, nuestro proyecto retraso el upgrade hasta tener un sistema estable. Una vez alcanzado ese hito se hicieron esfuerzos por migrar a ROS Hydro, una vez instalado el nuevo ambiente, todo funcionó de acuerdo a lo esperado. Luego, se realizó otra nueva actualización con la versión de ROS Jade, al igual que anteriormente, ningún problema fue reportado por parte del framework.

Finalmente, se logró desarrollar exitosamente el sistema de control para el observatorio Ckoirama, ubicado en el desierto de Atacama a 90km al sur de la ciudad de Antofagasta, para ello se implementó un sistema que considera la operación del domo (apertura y cierre), telescopio (monturas LosMandy G11 y Celestron) y clima, utilizando como interfaz una base de datos generada por Weewx obtenida desde una estación meteorológica Davis Vantage Pro 2.

Con el sistema desarrollado, el observatorio es capaz de dirigir el telescopio hacia un objetivo, abrir y cerrar el domo de forma remota o por factores climáticos (cierre por emergencia) y llevar la estadística del clima. El observatorio será energizado

utilizando energía solar, sin embargo, para efectos de desarrollo se realizaron pruebas utilizando un generador, puesto que el sistema solar aún no está concluido a la fecha.

La conectividad hacia el observatorio está provista por una señal de 3G utilizando un chip de celular y es distribuida en el observatorio, utilizando un router industrial. Al utilizar señal 3G, priorizar y optimizar el tráfico es una tarea crítica debido a que el tráfico está limitado. Es en este punto, donde ROS y Linux cumplen otro papel vital al permitir conexión vía comandos, lo que disminuye drásticamente el consumo de datos en comparación con otras soluciones como VNC, donde aparte del alto consumo que genera la transmisión de imágenes, las órdenes no suelen llegar con la precisión o velocidad que se desea, cosa que no sucede con comandos como los provistos por ROS y Linux.

CONCLUSIONES

Considerando la problemática presentada, se logró demostrar satisfactoriamente la hipótesis planteada al inicio del proyecto, usar ROS como framework para el desarrollo de un sistema de control y teleoperación del observatorio, logrando implementar exitosamente un prototipo de teleoperación para observatorios. De acuerdo a la literatura, la comunidad astronómica es reducida, distribuida y altamente especializada, por lo tanto, hacer uso de un framework ampliamente utilizado en el campo de la robótica, brinda una solución de software económica y altamente soportada y diseñada para el control de robots.

Otro aspecto importante a considerar es el soporte de software recibido por parte de ROS, en relación a los canales de comunicación disponibles para el framework como también a las constantes actualizaciones por parte del mismo la que fueron perfectamente aplicadas al observatorio, lo cual significa que los desarrollos realizados en el ámbito de la robótica fueron también aplicados en astronomía.

Actualmente la astronomía tiende a robotizar y automatizar los observatorios junto con la tendencia hacia estándares industriales, por lo que utilizar un software de robótica para acelerar ese proceso, es el camino más indicado. La evolución de los observatorios puede perfectamente ir de la mano con la evolución de la robótica y la teleoperación.

Finalmente, producto de la investigación realizada, se pudo determinar la estructura general de un observatorio astronómico, así como sus requerimientos de funcionamiento, éstos se encuentran generalmente alejados de las urbes, con el objetivo de eliminar o reducir cualquier interferencia que la vida humana pudiera generar, ya sea a nivel de radiofrecuencias como luminiscencia. De forma general, los observatorios están constituidos de una serie de aplicaciones cliente/servidor, aquellas que actúan como clientes son las que son utilizadas por los operadores para realizar las operaciones en el sistema, mientras que las que funcionan como servidores son los drivers asociados a cada dispositivo a controlar. Este comportamiento es la base de funcionamiento de ROS y también de la mayoría de los observatorios.

REFERENCIAS

- [1] G. Chiozzi *et al.*, “Enabling technologies and constraints for software sharing in large astronomy projects”. In *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70190Y-70190Y-12.
- [2] M. Quigley *et al.*, “ROS: an open-source Robot Operating System”, *Icra*, vol. 3, N° 1, p. 5, 2009.
- [3] F. Vilardell, G. Artigues, J. Sanz, Á. García-Piquer, J. Colomé, and I. Ribas, “Using Robotic Operating System (ROS) to control autonomous observatories”, 2016, vol. 9913, p. 99132V.
- [4] ROS, “ROS.org | Powering the world’s robots.” [Online]. Available: <http://www.ros.org/>.
- [5] J. Gates, W.T.S. Deich, A. Misch, and R. I. Kibrick, “Modern computer control for Lick Observatory telescopes,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70190C-70190C-12.
- [6] E. Allaert and G. Raffi, “Preparation for VLT software commissioning at Paranal,” in *Optical Science, Engineering and Instrumentation '97*, 1997, pp. 20-27.
- [7] G. Raffi and K. Wirenstrand, “VLT control software in its final test phase”, in *Optical Telescopes of Today and Tomorrow*, 1997, pp. 996-1004.
- [8] M.J. Kiekebusch, G. Chiozzi, J. Knudstrup, D. Popovic, and G. Zins, “Evolution of the VLT instrument control system toward industry standards”, in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77400T-77400T-12.
- [9] W. Pessemier *et al.*, “Towards a new Mercator Observatory Control System”, in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77403B-77403B-10.
- [10] G. Chiozzi *et al.*, “CORBA-based Common Software for the ALMA project”, in *Astronomical Telescopes and Instrumentation*, 2002, pp. 43-54.
- [11] T.-C. Shen *et al.*, “Development of telescope control system for the 50cm telescope of UC Observatory Santa Martina”, in *SPIE Astronomical Telescopes + Instrumentation*, 2012, p. 84511T.
- [12] P. Kubánek *et al.*, “Operating a global network of autonomous observatories,” in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77400U-77400U-12.