RevMexAA (Serie de Conferencias), **48**, 70–75 (2016)

ASTROBO: TOWARDS A NEW OBSERVATORY CONTROL SYSTEM FOR THE GARCHING OBSERVATORY 0.6M

T. Schweyer[1,2], P. Jarmatz[2], and V. Burwitz[1]

IV Workshop on Robotic Autonomous Observatories (Torremolinos, Málaga, Spain, September 28- October 2, 2015)
Editors: M. D. Caballero-García, S. B. Pandey, D. Hiriart & A. J. Castro-Tirado

RESUMEN

El recientemente instalado Telescopio del Observatorio del Campus de Garching (COG) de 0,6m de diámetro engloba un amplio conjunto de instrumentos, incluyendo uno de gran campo y una variedad de espectrógrafos. Para apoyar todos estos instrumentos y mejorar su uso temporal, hemos decidido desarrollar un nuevo sistema de control partiendo de cero, que podrá permitir observaciones tanto en modo autónomo como en modo manual (para prácticas de estudiantes). Está construido por medio de una arquitectura de servicios multi jerárquica, que permite comunicaciones bien especificadas entre sus componentes independientemente del lenguaje de programación usado. Su esquematización modular permite un prototipo rápido de sus componentes al igual que una rápida implementación de un complejo software de control de la instrumentación.

ABSTRACT

The recently installed Campus Observatory Garching (COG) 0.6m telescope features a wide array of instruments, including a wide-field imager and a variety of spectrographs. To support all these different instruments and improve time usage, it was decided to develop a new control system from scratch, that will be able to safely observe autonomously as well as manually (for student lab courses). It is built using an hierarchical microservice architecture, which allows well-specified communication between its components regardless of the programming language used. This modular design allows for fast prototyping of components as well as easy implementation of complex instrumentation control software.

Key Words: techniques: miscellaneous — telescopes

1. INTRODUCTION

As part of a joint effort of MPA, the high energy group of MPE[1], the TUM[2] and the Excellence Cluster Universe a small observatory was built. It was setup in April 2013 to provide students on the campus with a facility to do student lab courses, small science research projects and instrument development. It is located on the Garching research campus, which is about midway between Munich and Munich airport.

In the following we first describe the telescope hardware and observatory infrastructure (§ 2) This is followed by a short description of the requirements and philosophy that drove the design of Astrobo (§ 3). The architecture (§ 3.3) and the various subsystems (§ 5 – 9) are then described in greater detail. As a last point we discuss a few operational issues (§ 10).

[1]Max-Planck-Institut fuer extraterrestrische Physik, Giessenbachstrasse 1, 85748 Garching, Germany (welterde@mpe.mpg.de).

[2]Technische Universität München, Physik Dept., James-Franck-Str., 85748 Garching, Germany.

2. THE OBSERVATORY

The 4.5m Baader Allsky clamshell dome is located on the roof of the newly constructed MPA/MPE administration building and is thermally isolated from it. The dome can be controlled over a simple serial protocol over RS232 and has a built in UPS (Uninterruptible Power Supply) to automatically close itself in case of a power cut.

The telescope is an CDK24 (Corrected Dall-Kirkham) from Planewave, which has an aperture of 0.61m and an focal ratio of $f/6.5$. It is mounted on a GM4000 HPS mount from 10micron, which communicates with the control computer via Ethernet. The protocol is a fairly well documented extended version of LX200 also used by other telescope mounts.

The various instruments (see section 2.2) attach to the focuser and rotator unit IRF90 from Planewave, which is controlled over a RS232 port.

2.1. *Infrastructure*

All devices are connected on individual ports of network-enabled power distribution units (PDU), which support *SNMP* (Case et al. 1990) and are fed by the campus power grid. This allows for remote and automatic management of the power state of all relevant devices of the observatory.

IV Workshop on Robotic Autonomous Observatories (Torremolinos, Málaga, Spain, September 28- October 2, 2015)
Editors: M. D. Caballero-García, S. B. Pandey, D. Hiriart & A. J. Castro-Tirado

The Reinhardt MWS 9 weather station (with rain and cloud sensor) is attached to a pylon located next to the dome and supplies its sensor values every 2 seconds to an attached raspberry pi (small embedded computer) over a RS232 connection. As an added security measure the rain sensor has an additional connection to the dome controller, which will trigger an automatic closure of the dome.

In addition to the weather station we also have the Allsky-340 camera from SBIG.

Additional sensors and actuators, such as various temperature sensors, a secondary rain sensor, sky temperature sensor, focus motors, etc. are foreseen to be connected using Controller Area Network (CAN).

2.2. *Instruments*

GOWI (Generic Off-the-shelf Wide-field Imager) is a commercial SBIG STX-16803 camera with attached filter wheel, which features a ON Semiconductor KAF-16803 CCD. The CCD has 4096 by 4096 pixels of 9μm pixel size for a total field of view of 31 by 31 arcminutes. The camera also includes a ON Semiconductor KAI-0340 (640 by 480 of 7.4μm pixel size) as guiding CCD. The camera is attached to the control computer via Ethernet as well as USB.

The attached filter wheel FW5-STX has 5 slots for square 65mm filters, which contains Baader R, G, B, white light, and 7nm Hα filters with very high transmission (larger than 90%).

In addition to our main imaging camera, we have two spectrographs available: DADOS and BACHES. DADOS is a simple grating spectrograph with low resolution.BACHES (Avila et al. 2007; Kozłowski et al. 2014) is a slit-fed high-resolution echelle spectrograph.

Each spectrograph will have a Remote Calibration Unit (RCU) coupled to it via fiber that will provide ThAr and a flat field lamp.

A fibre-fed echelle-spectrograph called LECHES with much higher stability and slightly higher resolution than BACHES is currently being commissioned.

Also we are currently in the process of building a new GAM (Guide Acquisition Module) for OPTIMA (Kanbach et al. 2008), which is an existing photon-counting photometer of MPE.

Finally we are in the process of constructing a multiport adapter that will allow automated switching between up to 5 available instruments. Right now it is necessary to manually switch instruments by dismounting the old instrument and mounting the new instrument. It will consist of 4 movable flip mirrors that can either divert the light off 90° to the side or let it pass in the middle.

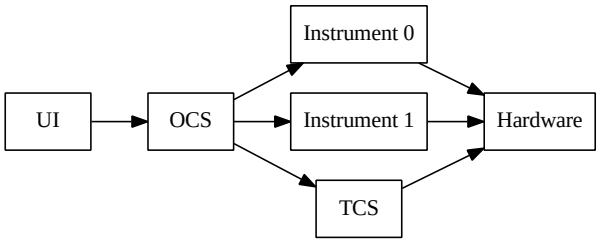

Fig. 1. Simplified System Architecture (see sec. 3.3)

3. DESIGN PHILOSOPHY

Major design requirements that were identified were: (a) Automated as well as powerful manual observing capabilities (b) Safe operations even in the presence or absence of a human operator (c) reusable and easily adaptable (other telescopes and new instrumentation)

3.1. *Modularity*

These days making the telescope software modular is an obvious point, but since there a multiple paths one can take to achieve this goal, it is worth mentioning regardless. The easiest approach would be to use just split the program into modules using whatever constructs the programming language used offers.

Our approach is to use multiple programs (daemons), which can be written in any programming language and have them communicate over network over a specified protocol. This way even distributing these daemons over multiple machines is easily possible.

A number of other existing control systems (Kubánek 2010; Stefanon et al. 2010; Nawrocki et al. 2010) also use this approach. However not all follow our approach of explicit protocol specification, subdivision of tasks, and multiple independent implementations as far as we do. The communication mechanism between the daemons is described in section 4.

3.2. *Safety*

Writing bug-free code is difficult. Thus our approach is two-fold: (a) keep the executed code minimal and as simple as possible (b) have redundant paths between sensor data collection and the hardware for especially safety critical systems.

In our case the dome is the only system that is extremely safety critical. The daemon **domed** that talks to the dome over a serial connection is extremely small and simple and thus fulfills requirement (a). But the higher-level daemons that take care of observatory operations are not as simple and

IV Workshop on Robotic Autonomous Observatories (Torremolinos, Málaga, Spain, September 28- October 2, 2015)
Editors: M. D. Caballero-García, S. B. Pandey, D. Hiriart & A. J. Castro-Tirado

thus much more prone to bugs. For this reason we have a secondary daemon (called the **safetyd**), which directly interfaces with the meteorologic sensors and the **domed**, and provides certain hard-limits, which cannot easily be overridden by a human operator.

3.3. *Architecture*

The overall architecture of Astrobo consists of several coupled subsystems that will be explained in greater detail in the following sections (see fig. 1). At the lowest are the drivers needed to interact with the actual hardware of the observatory (see sec. 5). Built on top of those drivers the telescope control system (TCS; see sec. 6) and the instrument subsystem (see sec. 7) are built. Explicit encapsulation of the telescope instruments is quite similar to how the ESO framework (Pozna et al. 2008) is structured. One layer up is the observatory control system (OCS; see sec. 8), which provides complex sequence execution (see sec. 8.1) and archival (see sec. 8.2), as well as managing the general state of the observatory. On top of the subsystem hierarchy the user interface subsystem (UI; see sec. 9) provides a homogeneous interface for manual observations as well as automated observations.

4. COMMUNICATION LAYER

At the lowest level ZeroMQ (Hintjens 2013) is used to handle message framing and certain messaging pattern, such as publish/subscribe and request/response.

Since ZeroMQ only deals with arbitrary binary messages we need to define a encoding format, for which we have chosen XDR (Eisler 2006), since it is standardized, has existing support in a number of programming languages and is fairly easy to implement if that is not the case.

For mapping between programming language native structures and the serial XDR stream we designed our own system which for now is called zrpc. The syntax of the zrpc protocol files is derived from Haskell with some modifications to make the syntax no longer whitespace-sensitive (see fig. 2 for example). It provides three functions: definition of type structures, interprocess messages (multicast) and request/reply commands. Commands have zero or more arguments, can return one value and can throw exceptions. Exceptions were deemed necessary to provide a way to have them fail early and provide information about the reason. By policy commands should return as soon as possible. For instance starting the exposure should not block until it has finished.

```
type Position = AngleDeg :: Double
              | DistanceMeter :: Double
              | RaDecDegJ2000 :: Double, Double

type State = InPosition :: Position
           | Moving :: Position, TargetPosition
           | Error :: String

type AbortMode = ResetPosition
               | KeepPosition

type Errors = InvalidPositionType
            | PositionOutOfBound
            | InvalidState

protocol AstroboMovement/42 version 1 where
    command move/1 :: Position -> Void throws Errors
    command abort/3 :: AbortMode -> Void throws Errors
    command getState/4 :: Void -> State

    message MoveStarted/1 :: Position
    message MoveCompleted/2 :: Position
    message MoveAborted/3 :: Position, AbortMode
    message ErrorOccurred/4 :: Void
end
```

Fig. 2. Shortened zrpc definition of the generic movement protocol

But the main difference to most evaluated alternatives (such as google protocol buffers, ROS messages, Apache Thrift, etc.) are the variant types, which are a common feature of type systems of functional programming languages (see **State** in fig. 2 for an example). They allow clutter-free description of complex states.

Another feature of zrpc is the integration of trace tags on all messages, which get propagated automatically using thread-local primitives. These allow much easier debugging of distributed systems, since typically it is difficult to correlate errors in one process on one machine with actions in another process on potentially another machine. This is a common problem of distributed architectures.

As we believe this data serialization system will be useful outside of Astrobo, it will be maintained outside of the Astrobo project and will be available separately. Right now we have two code generators, one written in C++ and one written in Haskell, that can generate code for three languages (C++, Python and OCaml). This allows easy verification if the implementation actually follows the specification.

Built on top of zrpc we have written a library called astrobo-net, which provides higher-level functionality for a few core protocols. Core protocols are the *variables protocol*, which provides a database of sensor values and changeable parameters, the *movement protocol* (see fig. 2) and the *safety protocol*.

The *safety protocol* is a simple broadcast of a timestamp-safe/unsafe tuple, which is sent every couple of seconds by the previously mentioned **safetyd** (see sec. 3.2). The library in this case only pro-

IV Workshop on Robotic Autonomous Observatories (Torremolinos, Málaga, Spain, September 28- October 2, 2015)
Editors: M. D. Caballero-García, S. B. Pandey, D. Hiriart & A. J. Castro-Tirado

vides a few convenience functions to correctly handle timeouts and absence of messages (in case of network interruption or the **safetyd** crashing).

The *variable protocol* provides a mechanism for a daemon to provide a set of changeable parameters, and read-only sensor and status values to other daemons. Daemons that have to change process variables of other daemons have to first open them in write mode. At this stage the daemon exporting the variable can identify conflicting requests (which with no explicit open/close semantic will be quite difficult to identify).

5. HARDWARE DRIVERS

An observatory contains a lot of hardware that needs to be managed. However, very rarely will different observatories have the exact same hardware components nor a completely different set.

Thus to make the software as reusable as possible, the hardware layer should be as modular as possible. There are different approaches to this problem: e.g. ASCOM (Denny 2002) provides an API that software can use to access hardware while being ignorant about finer details. This however has several drawbacks: (a) it is windows-only (b) poorly implemented (c) an API and not a network protocol (thus being limited to running on the same machine).

RTS2 (Kubánek 2010) solves this much better with a custom ASCII protocol. It probably also has the widest range of hardware support besides perhaps ASCOM. The protocol as it is implemented has certain drawbacks as well: (a) protocol not defined strictly (communication errors can occur at runtime) (b) it uses a database model for changing hardware parameters. The drawbacks of the database model are that it makes reasoning about interleaved states and reset states more difficult.

Our approach is to use a network protocol (see section 4) as well, but with a much more explicit and specific protocol specification than RTS2 does. For parameters that should not (and potentially cannot) be changed on short time scales (and thus don't suffer from the interleaving problem) we do however use the database model as well (see sec. 4). This is also true for sensor values, which cannot be changed. Our database equivalent is implemented using a hierarchical variable system, which allows daemons to export a certain set of variables, change them and subscribe to changes.

Two essential protocols for hardware interaction are the generic movement protocol and the camera protocol. The movement protocol (see fig. 2) is used to control the dome, focuser, rotator and telescope.

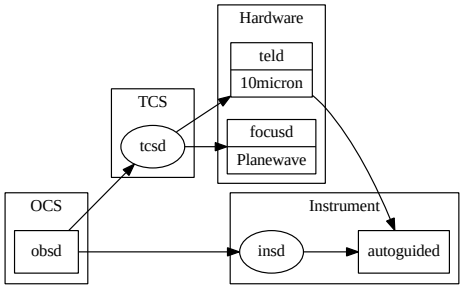

Fig. 3. Components of the TCS subsystem (see sec. 6 below)

6. TELESCOPE CONTROL

In addition to the various aspects of telescope control (pointing, tracking and guiding) the telescope control subsystem daemon (**tcsd**) also manages the lightbeam and focus (see fig. 3).

For imaging with GOWI (see 2.2) rough pointing on the order of a couple of arcminutes is usually good enough and easily achievable with our mount. For slit- and fiber-fed spectrographs as well as photometers however more precise object positioning is a must ($\approx$ 1 arcsecond). Blind pointing of the telescope is simply not good enough in this case for automated operation. Iteratively solving the astrometry and offsetting until convergence is the algorithm we intend to implement.

While short exposures up to 10 minutes are no problem with our current mount, guiding beyond that time scale is advised. This poses a special problem, since each instrument will need it's own guide camera and this somewhat breaks the instrument submodule hierarchy. Implementing guiding also requires a more realtime path between the guide camera and the telescope mount. An easy approach to this would be to have the autoguide daemon send messages to the telescope mount daemon. Our approach is to have the telescope mount daemon subscribe to a guide commands channel from the autoguide daemon (see fig. 3). This has the benefit of there ever only being one autoguided process the telescope listens to.

Light beam management involves controlling the multiport adapter (see sec. 2.2) to select the correct mirror configuration to divert the light towards the currently active instrument.

Focus is also an important issue for all instruments, since it strongly depends on temperature and perhaps weakly on position. To make best use of available observing time and get the best possible image quality, the dependance on these parameters should be correctly modeled and no refocusing dur-

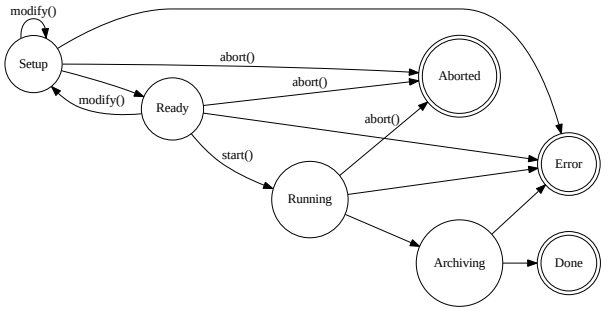

Fig. 4. State Machine of high-level instrument subsystem exposures (see sec. 7 below)

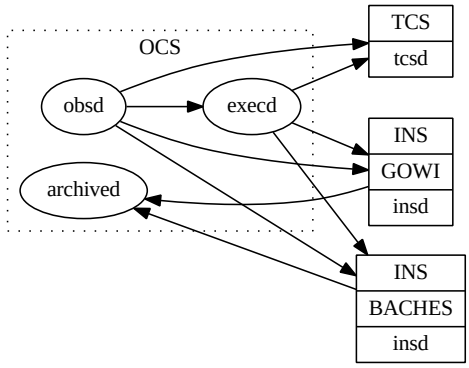

Fig. 5. Detail view of interactions of observatory subsystem components with other subsystems (see. sec. 8 below)

ing the night should be necessary.

7. INSTRUMENT SUBSYSTEM

Built on top of the hardware subsystem is the instrument subsystem, which provides certain high-level abstraction over the instrument-specific details. It is designed with the assumption that a instrument collects light from the sky for a certain duration of time and should thus be suitable for instrumentation ranging from simple cameras over photometers to multi-object spectrographs.

The main component/access point is a instrument-specific **insd** that takes care of all instrument internals, proper startup/shutdown and exposes a set of observation/exposure templates.

These templates expose a set of parameters, which can be modified for each specific exposure. This strikes the balance between complete encapsulation of the instrument and facilitating the writing of efficient observation sequences (see section 8.1 for more details on sequences).

Each exposure in terms of the instrument subsystem follows a certain state machine (see fig. 4), which allows one or more exposure to be setup while the previous one is still running (say changing the filter while the chip is being read out or exposing while downloading the image). An exposure is considered to be running until the data has completely arrived in the memory of the camera driver. Also of note is the fact that an exposure can be aborted at any stage except during the archival stage. This is a common requirement (Shortridge & Farrell 2004) to ensure that the instrument is able to observe again as soon as possible in case some unforseen event occurs.

8. OBSERVATORY SUBSYSTEM

The observatory daemon (**obsd**) takes care of dome, instrumentation and telescope state. For the dome it ensures the environmental soft-limits (such as cloud coverage, humidity, wind or sun altitude) are always met and closes the dome if they are not. It also monitors the state of all active instruments.

In the afternoon it first does the instrument startup and then decides when to open the dome based on sun altitude and M1 to outside temperature difference. During the twilights it executes the calibration OB (Observing Block; see sec. 8.1) to obtain twilight flats.

8.1. *Sequence Execution*

The execution daemon (**execd**) provides higher levels of the software with powerful observing blocks that can be scheduled as one unit. The observing blocks are not limited to just using one instrument and are also used for more complex tasks, such as focusing.

The first prototype of this execution daemon is written in and executes python scripts. These OB scripts provide a external structure that can be modified just before executing similar to ESO BOB (Allaert 2007).

8.2. *Archival*

A single **archived** (archive daemon) is subscribed to all exposure data channels of all data collecting processes. On receiving a new exposure it queries the **sensord** (sensor daemon) for all sensor values between the start and end time of the exposure (see fig. 6). The exposure data as well all environment sensor values are stored in one HDF5 (Folk et al. 1999) file. Multiple exposures forming one OB can also be easily stored in one file.

9. UI SUBSYSTEM

The top-most layer is the user-facing layer. This is however not quite true, since special daemons like GCN (Gamma-ray Coordinates Network; Barthelmy

IV Workshop on Robotic Autonomous Observatories (Torremolinos, Málaga, Spain, September 28- October 2, 2015)
Editors: M. D. Caballero-García, S. B. Pandey, D. Hiriart & A. J. Castro-Tirado

IV Workshop on Robotic Autonomous Observatories (Torremolinos, Málaga, Spain, September 28- October 2, 2015)
Editors: M. D. Caballero-García, S. B. Pandey, D. Hiriart & A. J. Castro-Tirado

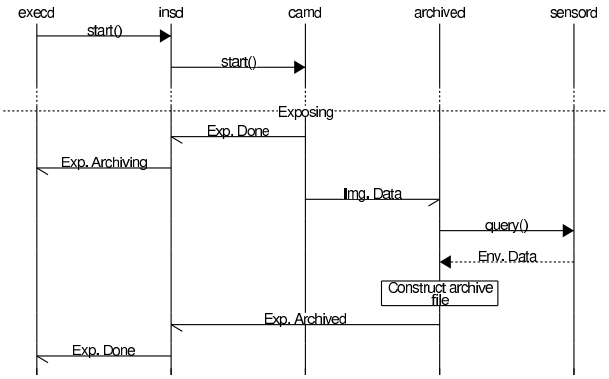

Fig. 6. Sequence diagram of one exposure of a typical observation including the archival process of image and environmental data

2008) or VOEvent (Williams & Seaman 2006) handling will also be implemented using the *UI protocol*. This gives a much more elegant overall design, since one only has to implement one method for allowing a user with higher priority to interrupt another user. No special handling of GRB observations or transient observations in general is required. Special cases of even higher priority, such as occultations, can be handled trivially this way.

Moving the knowledge of proposal priorities so far to the edge in the architecture does come with a price. Service Queues can only be implemented at this level of the overall architecture. Thus the UI daemon (**uid**) will quite likely be quite complex compared to all other daemons in the system.

Since this daemon will be exposed to the outside world, extra care has to be taken to ensure the safety of the network communication (use of TLS for instance) and that bugs in the daemon implementation stays contained (no buffer overflows, etc.). Thus using a memory-safe programming language is pretty much required.

10. OPERATIONS

Operations in the context of the control system essentially means installing the software on all relevant machines, configuring it, running all necessary components and dependencies and handling faults.

Since all linux distributions come with their own package manager, manually installing Astrobo is not considered a viable option for long-term maintainability. Thus packaging all components at least for Debian is a high priority for this project. These days configuration is no longer a major concern with a whole slew of different configuration management tools on the market (**Puppet**, **Ansible**, etc.). Process supervision is also a solved problem with software like **supervisord**, **consul**, **runit** or even **systemd**.

Monitoring is also a solved problem for the most part in the system administrator world with tools such as **Nagios** or **Icinga**. Centralized data logging can also be achieved using standard protocols such as syslog.

Our aim is to leverage as many industry-standard tools as possible to enable normal system administrators to maintain the system without requiring too much domain-specific knowledge. Additionally not reinventing the wheel allows us to focus on more important aspects of the control software.

REFERENCES

Allaert, E. 2007, 6th edn., European Southern Observatory, doc. No. VLT-MAN-ESO-17220-1332
Avila, G., Burwitz, V., Guirao, C., Rodriguez, J., Shida, R., & Baade, D. 2007, The Messenger, 129, 62
Barthelmy, S. 2008, AN, 329, 340
Case, J. D., Fedor, M., Schoffstall, M. L., & Davin, J. R. 1990, STD 15, RFC Editor
Denny, R. B. 2002, SASS, 21, 39
Eisler, M. 2006, STD 67, RFC Editor
Folk, M., Cheng, A., & Yates, K. 1999, in Proceedings of Supercomputing, Vol. 99, 5–33
Hintjens, P. 2013, ZeroMQ: The Guide
Kanbach, G., Stefanescu, A., Duscha, S., Mühlegger, M., Schrey, F., Steinle, H., Slowikowska, A., & Spruit, H. 2008, in ASSL, Vol. 351, 153
Kozłowski, S. K., Konacki, M., Ratajczak, M., Sybilski, P., Pawłaszek, R. K., & Hełminiak, K. G. 2014, MNRAS, 443, 158
Kubánek, P. 2010, AdAst, 2010, 902484
Nawrocki, K., Ptasińska, M., Sokołowski, M., Użycki, J., & Zaremba, M. 2010, AdAst, 2010, 560378
Pozna, E., Zins, G., Santin, P., & Beard, S. 2008, in SPIE Conference Series, Vol. 7019, Advanced Software and Control for Astronomy II, 70190Q
Shortridge, K. & Farrell, T. J. Proc. SPIE, Vol. 5496, 50–56
Stefanon, M., Covino, S., Fugazza, D., Molinari, E., D'Alessio, F., Malaspina, G., Nicastro, L., Testa, V., Tosti, G., & Vitali, F. 2010, AdAst, 2010, 976890
Williams, R. D. & Seaman, R. ADASS XV, ed. C. Gabriel, C. Arviset, D. Ponz, & S. Enrique, 637