View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Isik University Academic Open Access

Agile development approach for the observatory control software of the DAG 4m telescope

B. Bülent Güçsav*[a], Deniz Çoker[a], Cahit Yeşilyaprak[a], Onur Keskin[b], Lorenzo Zago[c], Sinan K. Yerli[d]
[a]Atatürk Univ., Astrophysics Research and Application Center (ATASAM), Erzurum, Turkey; [b]FMV Işık Univ., Engineering Faculty, Dept. of Mechanical Eng., Istanbul, Turkey; [c]Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud, (HEIG-VD), Yverdon-les-Bains/Switzerland; [d]Orta Doğu Teknik Üniversitesi (METU), Physics Department, Ankara, Turkey.

ABSTRACT

Observatory Control Software for the upcoming 4m infrared telescope of DAG (Eastern Anatolian Observatory in Turkish) is in the beginning of its lifecycle. After the process of elicitation-validation of the initial requirements, we have been focused on preparation of a rapid conceptual design not only to see the big picture of the system but also to clarify the further development methodology. The existing preliminary designs for both software (including TCS and active optics control system) and hardware shall be presented here in brief to exploit the challenges the DAG software team has been facing with. The potential benefits of an agile approach for the development will be discussed depending on the published experience of the community and on the resources available to us.

Keywords: DAG, OCS, software development, OPC-UA, command line interface, 4-m telescope

1. OBSERVATORY CONTROL SYSTEM (OCS) SOFTWARE

The OCS software provides the enterprise level control of the DAG 4 mt class telescope and Focal Plane Instruments (FPI) and the observatory support systems. Following the requirements given in the Software Requirements Specifications for Observatory Control System[1] document draft and a series of meetings with DAG project team, three distinctive design constraints have been set initially.

2. DESIGN CONSTRAINTS

At first glance, the major constraints that should apply to a preliminary design is as follows :

- Programmable control and monitoring interface : The extent of the term programming describes an interface that makes an user to run automated observations from one or more Command Line Interfaces (CLI). Additionally, a batch-command execution environment will be available for sequenced jobs.
- Concurrent users and simultaneous subsystems use : The OCS software should support multiple user logins. Additional concurrency for multiple device/instrument access is also evident.
- Configuration Management : Run time configuration management for operational parameters as well as the behavior of the running software (i.e. it shall not be required to deploy the software on each configuration) should be present.

*gucsav@atasam.atauni.edu.tr; phone 90 442 236 3144, atasam.atauni.edu.tr

Software and Cyberinfrastructure for Astronomy III, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 991333 · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2234394

3. THE LIST OF EXISTING AND CANDIDATE SUBSYSTEMS

Service oriented approach as an initial choice of architecture, namely the industry standard OPC-UA[2], has been chosen for the low-level system access (Figure 1.) In this respect, we define a subsystem as any physical or logical system that is accessible via OPC-UA connections. We have also classified the subsystems into two groups, primary (TCS[3], aOCS[4], ECS[5], WFS) and the rest (for ICS, SEMS), based on the control domain for specific type users (sessions). The list of subsystems (already existing and foreseen) can be given as:

- Telescope Control System (TCS) : It is composed of the Mount Control System (MCS), the Telescope Auxiliary and Monitoring System (TAMS) and the Telescope Supervisor (TSV). TAMS is responsible for the safety of the system at all and low-level controls such as Hydro-static Bearing System (HBCS), primary mirror cover and telemetries.
- active Optics Control System (aOCS) : Active optics controllers and computers for the mirrors.
- Enclosure Control System (ECS) : The rotating enclosure controller and computers.
- Instrument Control System (ICS) : The control domain of this subsystem includes Adaptive Optics Control System (AdOCS) and six FPI (at most) that can physically be arranged on two Nasmyth platforms. At this early stage, it seems that it would be beneficial to separate the ICS as a subsystem. This way, a standard approach could be followed in accessing the low-level systems which will make the implementations a lot easier. The responsibility domain of this subsystem is to provide connectivity between the OCS and the various FPI and the necessary services for the OCS software. Following the definition given for the term subsystem, the OCS-ICS communication will use OPC-UA stack. For the other part of the interface, various connection mechanisms might exist that link the ICS to a particular FPI. Aside from mentioned above, science data header generation, depending on the particular FPI, will also be handled by the ICS. The science data header shall include : Instrument specific information, SEMS data for the particular observation or night, provided time information, primary subsystem information, session info.
- Wave Front Sensor (WFS) : It requires to be a separate subsystem as it is going to be responsible for the high order control[6] of the system as a whole.
- Site and Environment Monitoring System (SEMS)[7] :
    - Atmospheric Quality Measurements : Various measurements will provide atmospheric quality data on a regular basis.
    - Meteorological : Weather information data coming from various sources such as conventional meteorological devices and satellites. Collected weather data is going to be processed and will be sent to the OCS software as weather forecast information.
    - Seismic : A system that collects seismic activity data from the stations around the DAG site.
    - Observation Site Facilities : Infrastructures (power, communication, etc), surveillance and other support facilities.

4. USERS AND SESSIONS

One of the key concepts of the initial architecture is a *session.* A user of a session can interact with the nodes from the particular subsystems and the OCS internal components assigned to its control domain.

A session is created on user login from a GUI or a CLI and it can be uniquely described with the following four identifiers: Username, Password, Observing Run Id and User Type Request.

These user provided information shall be cross-checked from the Proposal Unit and Scheduler outputs. A session, depending on the identifiers, uniquely describes the node space partitioning of the available resources for the particular user.

Following list shows the user types and their authorized actions :

- WATCHCAT (or SYSTEM USER)
  - Can monitor all subsystems and OCS components.
  - Can reach to human beings via some conventional methods (i.e. email, SMS, push notification, physical LEDs and alarms.)
- OPERATOR
  - Can control primary subsystems (send commands).
  - Can control rest of the subsystems other than the primary.
  - Can monitor all of the subsystems.
  - Can send HIGH PRIORITY RANKED commands
  - There can be ONE operator at a time.
- NON-OPERATOR
  - Can control rest of the subsystems other than the primary and the FPI under operator usage.
  - Can monitor all of the subsystems.
  - Can send LOW PRIORITY RANKED commands
  - There can be MULTIPLE non-operators at a time. (Maximum number of non-operators are restricted to the available resources.)
- VIEWER
  - Can monitor all of the subsystems.
  - Not able to send commands.
  - There can be MULTIPLE viewers at a time. (Maximum number of viewers are restricted to the available physical resources (system + network load).)
- ENGINEER
  - Possesses all the capabilities of the OPERATOR as well as administration (eg. cancellation of active sessions, system state changes, application of predefined procedures, etc.)

These are OCS internal users. They can either directly be mapped to real world actors (i.e. human beings) or to some sort of robots that make the observations, calibrations, etc. on behalf of a real world actor. These mappings might or might not be in one to one correspondence with the actors specified in the Operations Concept Document (OCD)[8].

The given *priority ranking of commands* in the list above defines the prioritization of one session over another.

5. OCS INTERNALS (IMPLEMENTATION)

The main idea is -in some sense- to provide a versatile intermediate service layer (or a service network) to the higher level systems such as Scheduler, post-processing scripts, batch command execution environment, etc..

It is foreseen that the OCS software would mainly be developed on an application server. All the core functionality is executed by the components that collectively form a *core services framework* of the OCS. Inter component communications will be handled by the mechanisms (container services, messaging API) provided by the application server. The SEMS and the ICS are exceptions in this architectural layout as they reside outside of the application server. Their proximity to the low-level devices (and services), led us to think their implementations as standalone OPC-UA servers. In order to be aware of the rest of the subsystems in real time, these OPC-UA servers are going to act as OPC-UA clients as well (i.e. essentially for monitoring via subscriptions).

Each component family in the application server domain will implement an OPC-UA client per se to perform the necessary monitoring directly. This way it will enable us to give the power of pub-sub mechanism of OPC-UA architecture independently to each component. This approach could be given up depending on the possibility of achieving the soft real-time access requirements of the subsystems via the messaging API (i.e. the greatest common denominator of the minimum sampling intervals of all subscribed nodes is maintained by messaging API when the communication network is at full load.)

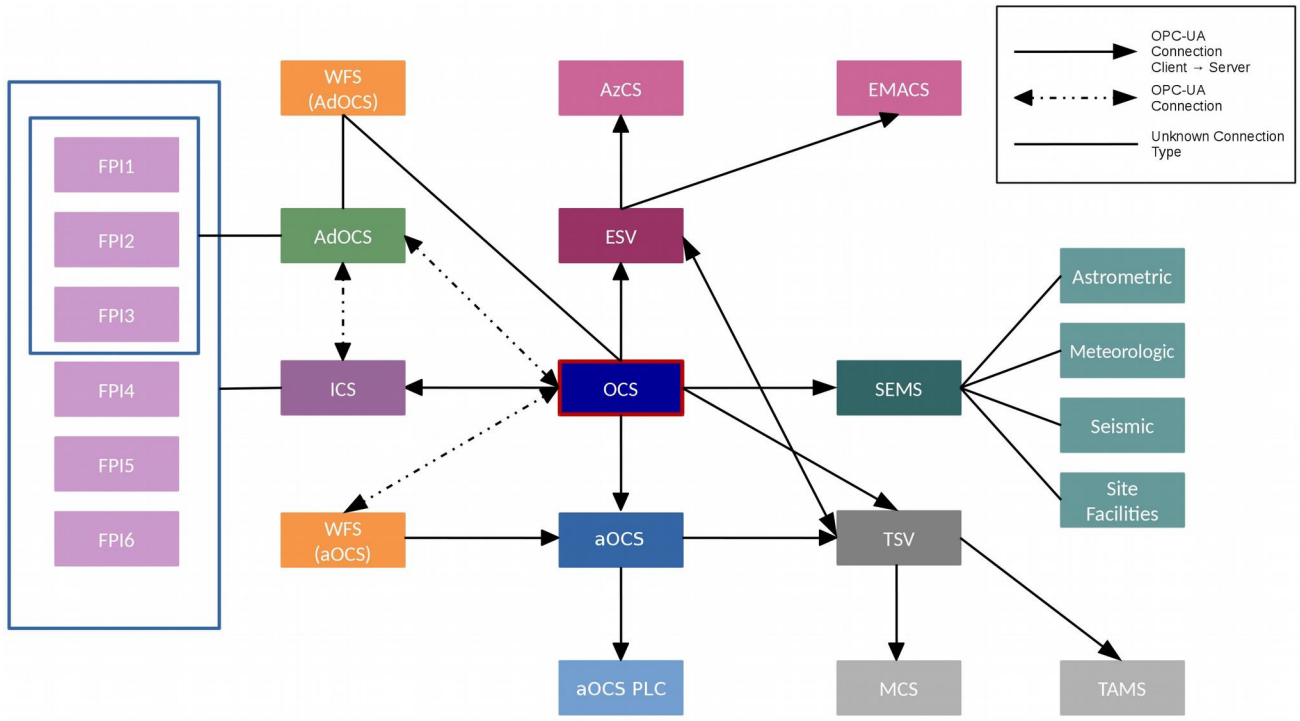

Figure 1. OPC-UA Client-Server layout of the subsystems. Directions of the arrows show the client-server relationships. Not yet known connections are shown without arrowheads.

5.1 Command Line Interface (CLI)

Instead of implementing a brand new command line interface for our own purposes, one viable option is to use a widely accepted and powerful tool already available. The inherent nature of the requirement has motivated us to think an interpreted programming language. The diversity of the available scientific frameworks makes Python environment a suitable candidate. Considering the approach we have developed so far, following issues must be addressed :

- *Direct* commanding of the subsystems should be available through CLI.
- CLI should provide a programmable interface.
- Users should be able to login through CLI (as well as GUI) which triggers a session start.
- Simultaneous usage of more than one CLI for a particular session (as well as different sessions) shall be supported.
- There should be an OPC-UA stack and Software Development Kit (SDK) available for the programming environment or at least by using the libraries and bindings available, a programmer could be able to control and monitor the subsystems (via some sort of messaging mechanisms) within a script/program.

5.2 Graphical User Interface (GUI)

These family of components shall provide the necessary support for a range of graphical ways of accessibility to the OCS software. For the classical mode observations, these components are service points for graphical clients which collectively form the primary source of control and monitor of the system as a whole. In parallel to the classical mode, these interfaces will provide a helper mechanism for the automated ways of operations. Web based accessibility is also a matter of concern by means of the provided interfaces.

5.3 Interfaces to the higher level systems

According to the OCD two types of observation are defined; Classical Mode and Queue Based Mode. In order to address the complexity of the queue based mode, an interface class of components will be defined between the Scheduler and the OCS. The Proposal Unit is another high level system that should be interfaced within the OCS software.

5.4 Data Handler

A class of components providing the functionality of data storage and access should be implemented by considering the factors stated below:

- A centralized logging system for both subsystems and OCS internals.
- Storage and accessibility of the ICS produced science data.
- Support for user configurable data formats for science data.
- Various forms of access to the stored data through CLI and GUI.
- An environment for pre-processing and post-processing of science data.

- Safe replication of all of the data to the tier 2 (ATASAM) storage.

In order to avoid the single point of failure, the services providing the above functionality shall be implemented distributively.

5.5 Resource Management across the OCS software

The union (logical aggregate) of all subsystems' OPC-UA address space nodes is called node space. It defines all the control and monitor domain for low-level systems of the OCS software. Different equivalence relations (with respect to the type of a session, allowed cross-subsystem conditions etc.) might be defined on the node space where each relation simply means a partitioning of the space. The complexity of concurrency can be handled by selecting the most efficient partition described by the conditions (Figure 2.)

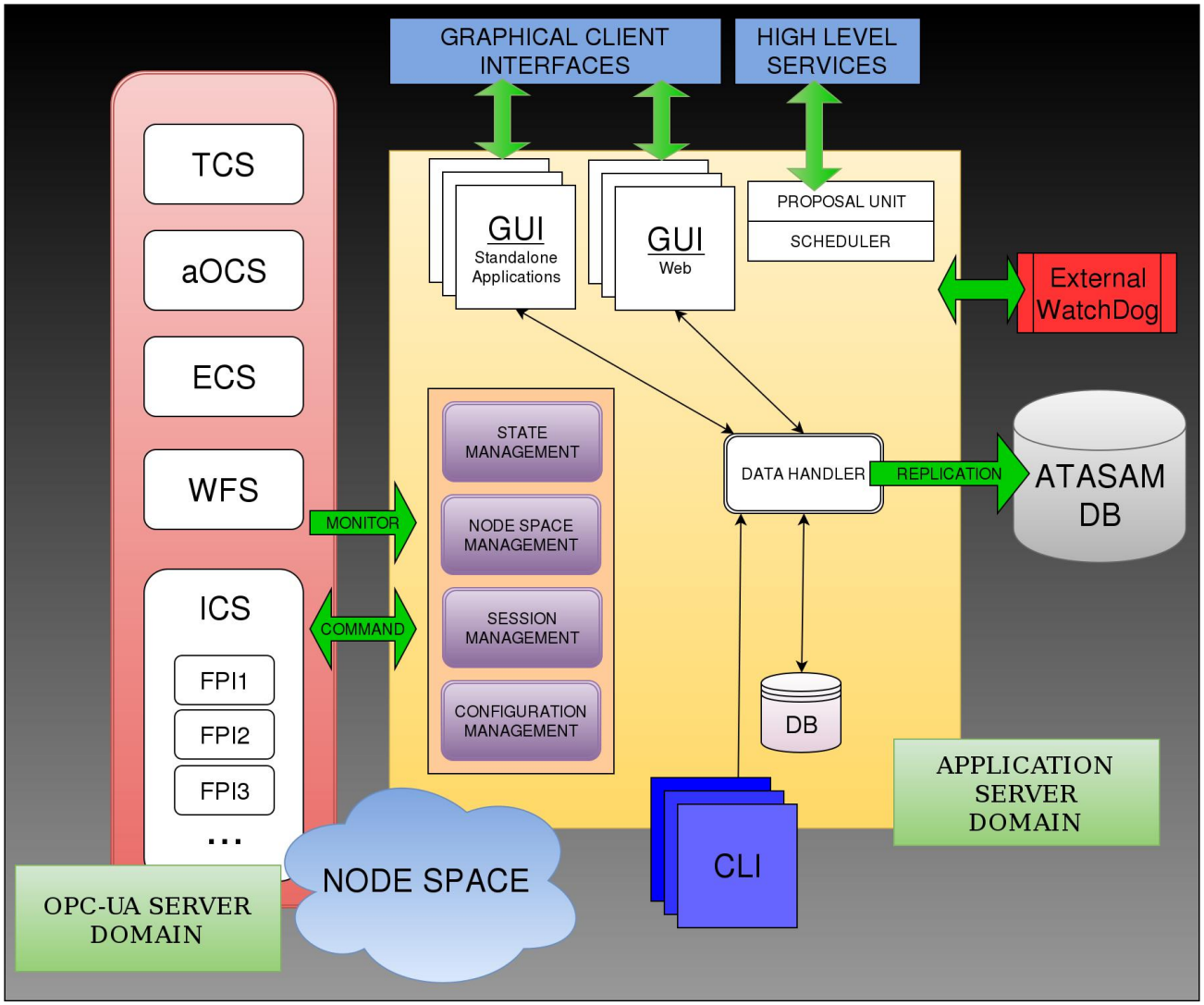

Figure 2. OCS Internals – Logical Layout.

Depending on the ideas developed so far it is possible to define -at least- some vague boundaries for the core services framework. This grouping of services (and the actions taken place) include, session management (handles the requests from the GUI and the CLI, makes authentications and authorizes the user of a session by interacting with the proposal unit) , node management (partitions the node space on request and assigns the demanded nodes to a session), configuration management (apply configurations that might apply to the current state and watch configuration update requests for a session or subsystem), and state management (coordinates the OCS and the subsystem states, constantly checks the in house components and communicates with the external watchdog, executes the routine procedures defined below.)

There shall be well defined procedures for routine actions such as start-up, safe shutdown and safety-emergency.

- Start-up procedure : Successful completion of this procedure implies that all internal components are alive and the internal communication channels are active. Furthermore, it is assured that all of the node space is accessible.
- Shutdown : The only reason for existence of this procedure is to test the OCS software working. It describes a sequence of steps for safe shutdown.
- Safety/Emergency : Although the failing subsystem will not be accessible from the OCS, bringing the rest of the system to a safe state is the responsibility of the OCS software.

At least the following four OCS internal states seems apparent :

- Fault : This state is ONLY triggered by an OCS internal reason (a software fault). “OCS IN FAULT STATE” does NOT imply that individual components which are already in operational state will stop working.
- Intermediate state (Idle or Waiting) : OCS is waiting for synchronization with the subsystems, or some internal condition prevents the OCS to enter into the Operational state.
- Operational : If the system is not in any other state, this means the OCS is operational.
- Engineering : Any subsystem in fault state will bring the OPC-UA connection between the relevant subsystem and the OCS down automatically. This situation requires human intervention for the faulty subsystem (i.e. the subsystem can be reached via engineering GUI), whilst the rest of the OCS should be operational/accessible.

Each component knows the current OCS-internal state. These OCS-internal states might also represent (indirectly) the particular subsystems' states. i.e. During the start-up sequence of each component, interlocks and states of the particular subsystems will automatically be checked via OPC-UA subscriptions.

6. BEGINNING OF THE NEXT PHASE

In order to able to reach a conclusion about the preliminary architecture , we have to be sure about the following items by taking immediate actions (i.e. by prototyping)

- Solid understanding of the implementation details of the hypothetical CLI is crucial where the rapid prototyping plays an important role. To be more specific, the CLI integration with the core services must be checked.
- Although the interactions between the low-level systems is handled by OPC-UA, the heterogeneity in the implementation styles of the address spaces of different subsystems, push us to define an abstract common

interface that mitigates the burden of coding difficulties. This abstraction layer might also give a better opportunity to implement any OPC-UA address space of a future instrument/system.

ACKNOWLEDGEMENTS

Authors would like to thank Atatürk University, Erzurum/Turkey (Project No: 2011K120230); Atatürk University, Erzurum/Turkey, Astrophysical Research and Application Center (ATASAM), Erzurum/Turkey; Republic of Turkey, Ministry of Development; Orta Doğu Teknik Üniversitesi, Ankara/Turkey; Işık University, Istanbul/Turkey; Işık University, Center of Optomechatronics Application and Research (OPAM), Istanbul/Turkey; Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD), Yverdon-les-Bains/Switzerland for their support through out the DAG project.

REFERENCES

[1] Gucsav, B. B., Coker, D., Yesilyaprak, C., Zago, L., "Software Requirements Specifications," DAG internal documentation (2016)
[2] OPC Foundation, "OPC Unified Architecture Specification," <https://opcfoundation.org/developer-tools/specifications-unified-architecture> (June 2016).
[3] ATASAM, "TCS Design Report," (2016).
[4] ATASAM, "active Optics Control System Design Report," (2016).
[5] ATASAM, "Rotating Enclosure Control System Design Report," (2016).
[6] Filgueira, J. M., Bec, M., Liu, N., Peng, C., Soto, J., "End-to-end observatory software modeling using domain specific languages," Proc. SPIE 9152, (2014).
[7] Sandrock, S., Amestica, R., Duhoux, P., Navarrete, J., Sarazin, M. S., "VLT Astronomical Site Monitor: Control, Automation, and Data Flow," Proc. SPIE 4009, 338-349(2011).
[8] Coker, D., Gucsav, B. B., "Operations Concept Document," DAG internal documentation (2016)