Preliminary ACTL-SLOW Design in the ACS and OPC-UA context

G. Tosti

(19/04/2016)

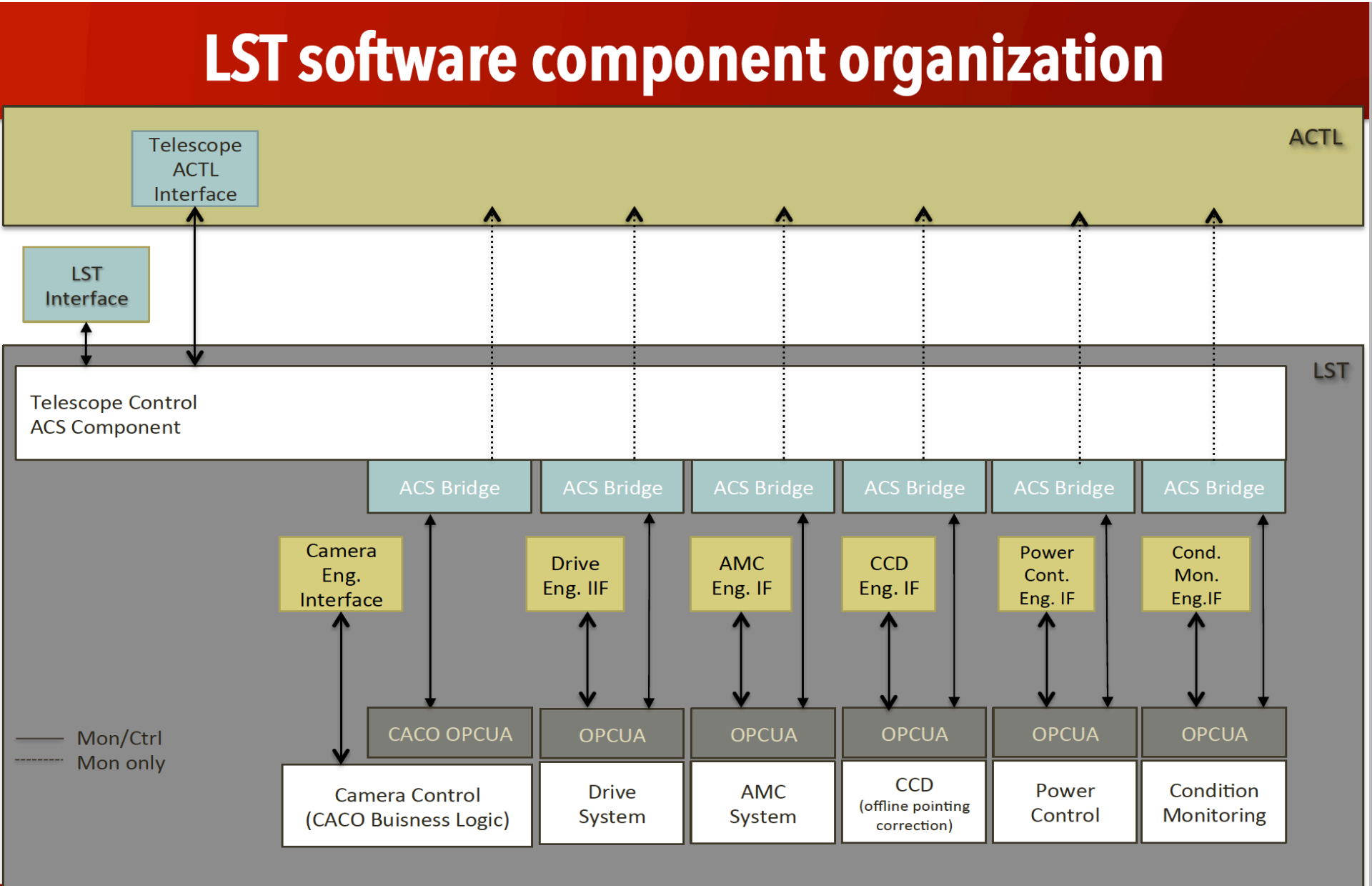
Summary

- General Introduction to ACS
- Preliminary ACTL-SLOW proposed design
- Hardware device integration in ACS and ACTL-SLOW
- Monitor and Control Points:
  - How to define them in an uniform way
  - Naming Convention

Modification to what is reported here could happen based on
the ongoing definition of the ACTL Architecture and of the ACTL-OPS /ACTL-SLOW interface.

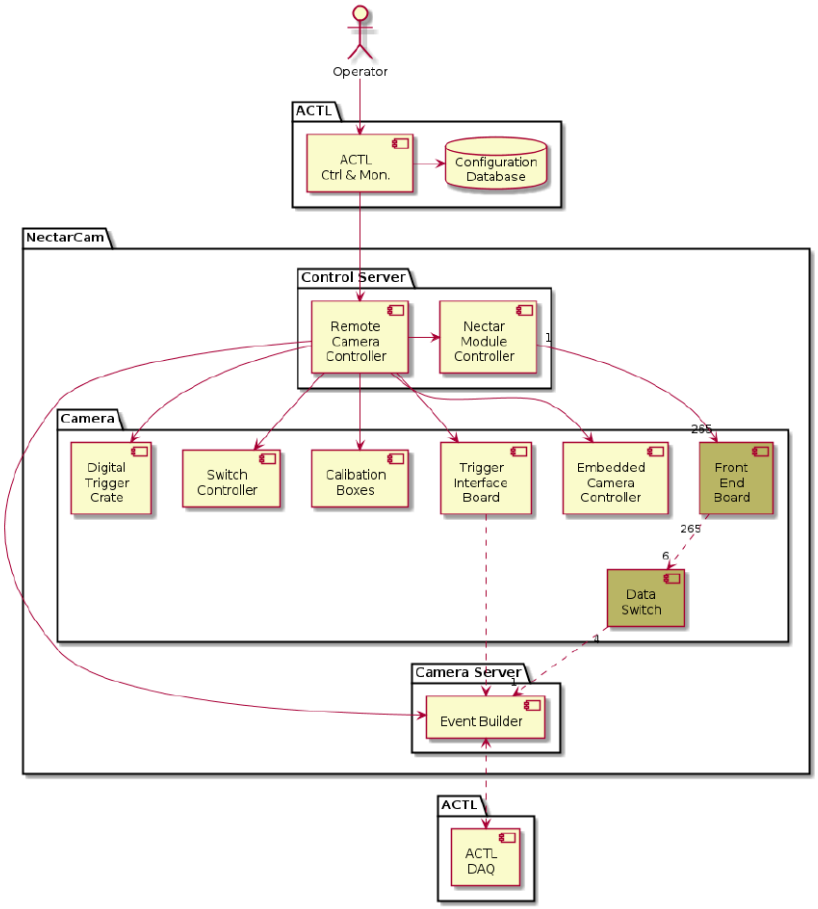
All reported here will be then formalized in ACTL-SLOW the design
document that will be a live document shared with all telescope teams.

Take advantages from diversity



Take advantages from diversity

Camera Slow Control functional model



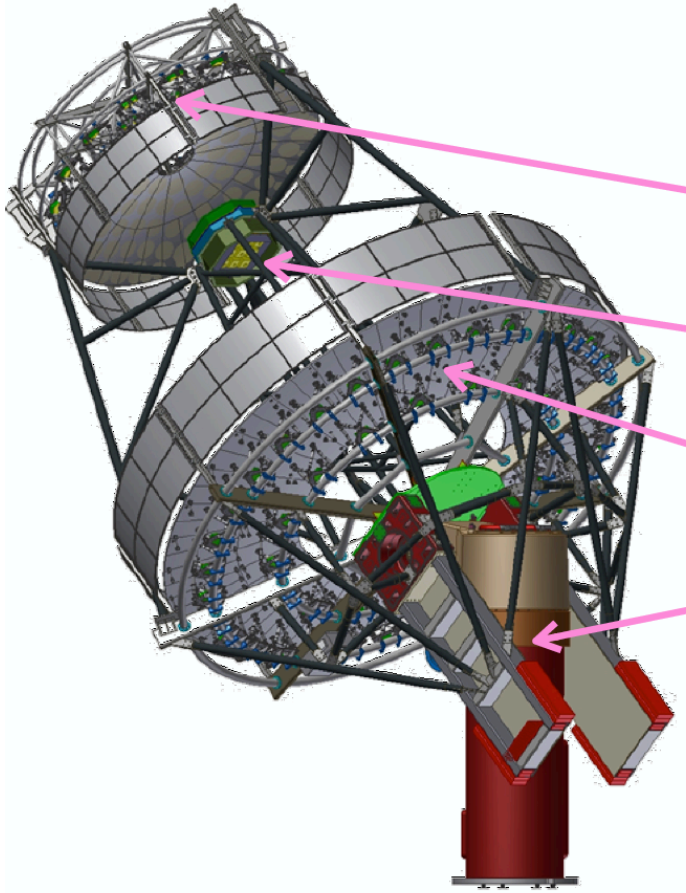
Take advantages from diversity

Software

	OPC UA server	Bridge	Control	Highler level control	Remarks
Drive system	Exists (in a PLC)	Exists	Python client in ACS	Scripts	PLC software also used at pSCT/SST-1M
CCD cameras	Exists (for Prosilica) on a SL machine	Exists	Java ACS Components	Scripts	Elaborate display software. No OPC UA server for Apogee yet.
AMC	Exists	Partly		Scripts	

Take advantages from diversity

pSCT subsystems



- Global alignment system
- Calibration and auxiliary systems
- Camera Slow Control
- Panel-to-panel alignment system
- Positioner control
- Array trigger and timing demonstrator

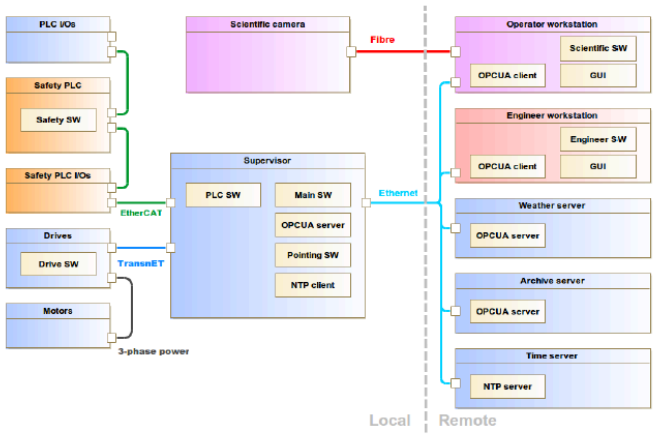
Custom solution mostly based on LabView

Take advantages from diversity

Telescope Control System

- Two different systems, one for drives and one for safety, with distinct modes of communication and developed in Beckhoff environment
- Hardware: COTS modules from Beckhoff
- OPCUA protocol as communication interface
- A local control box allows direct command of the motors close to the telescope

Software architecture
(Feb 2016)

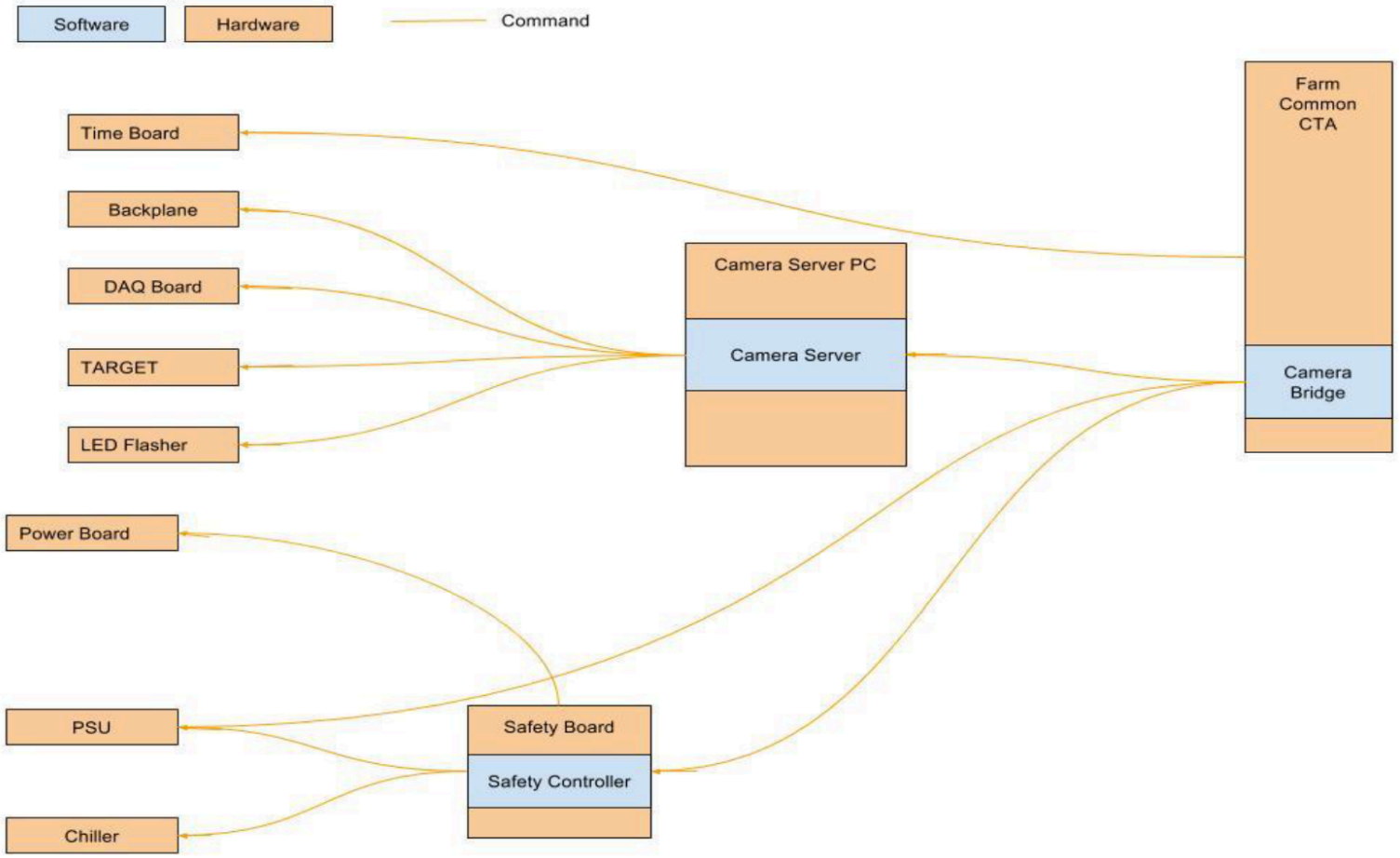


Take advantages from diversity

Schedule: What's next

Redesign of the HW – FW driven by prototyping lesson learnt

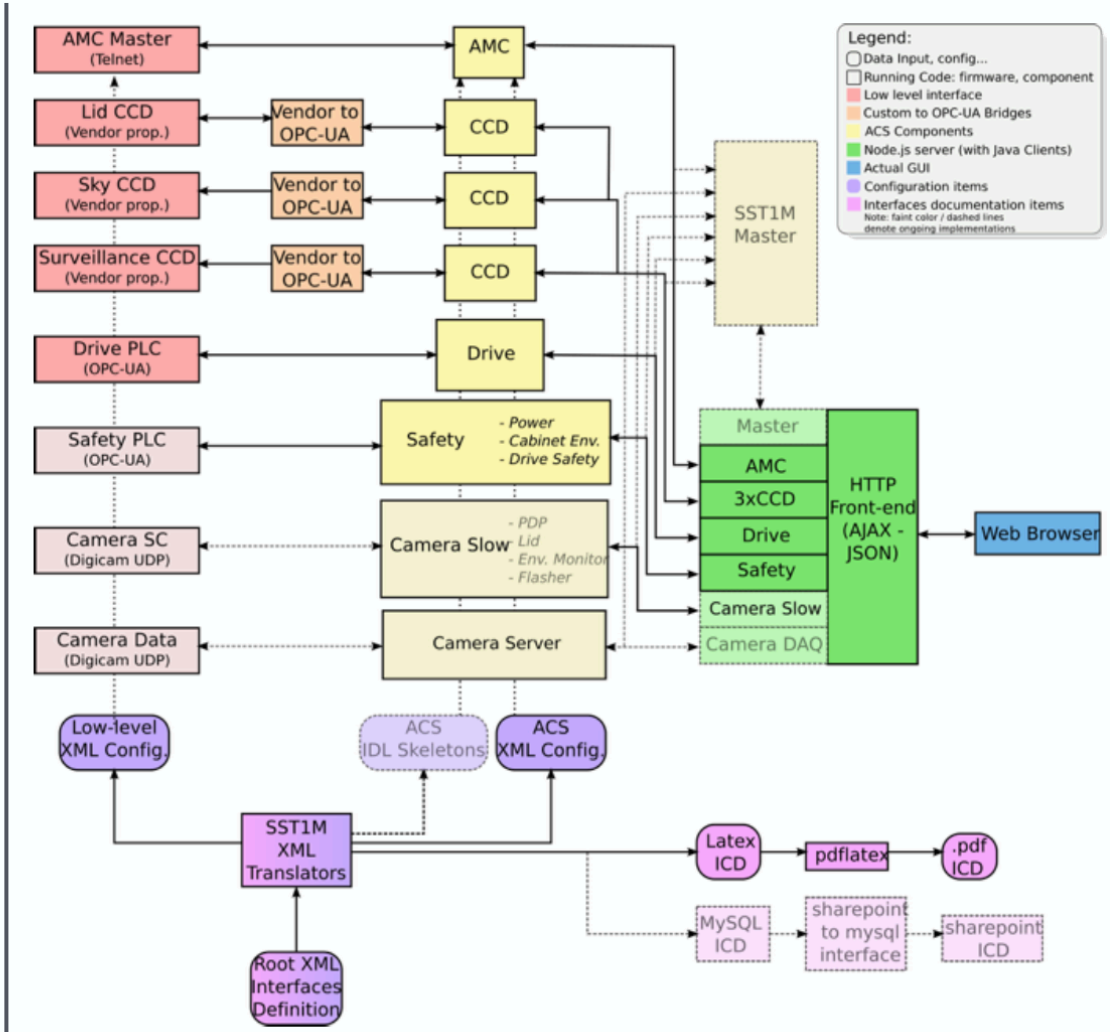
→ SW design we have in mind to reflect the changes



Take advantages from diversity

General Software Design

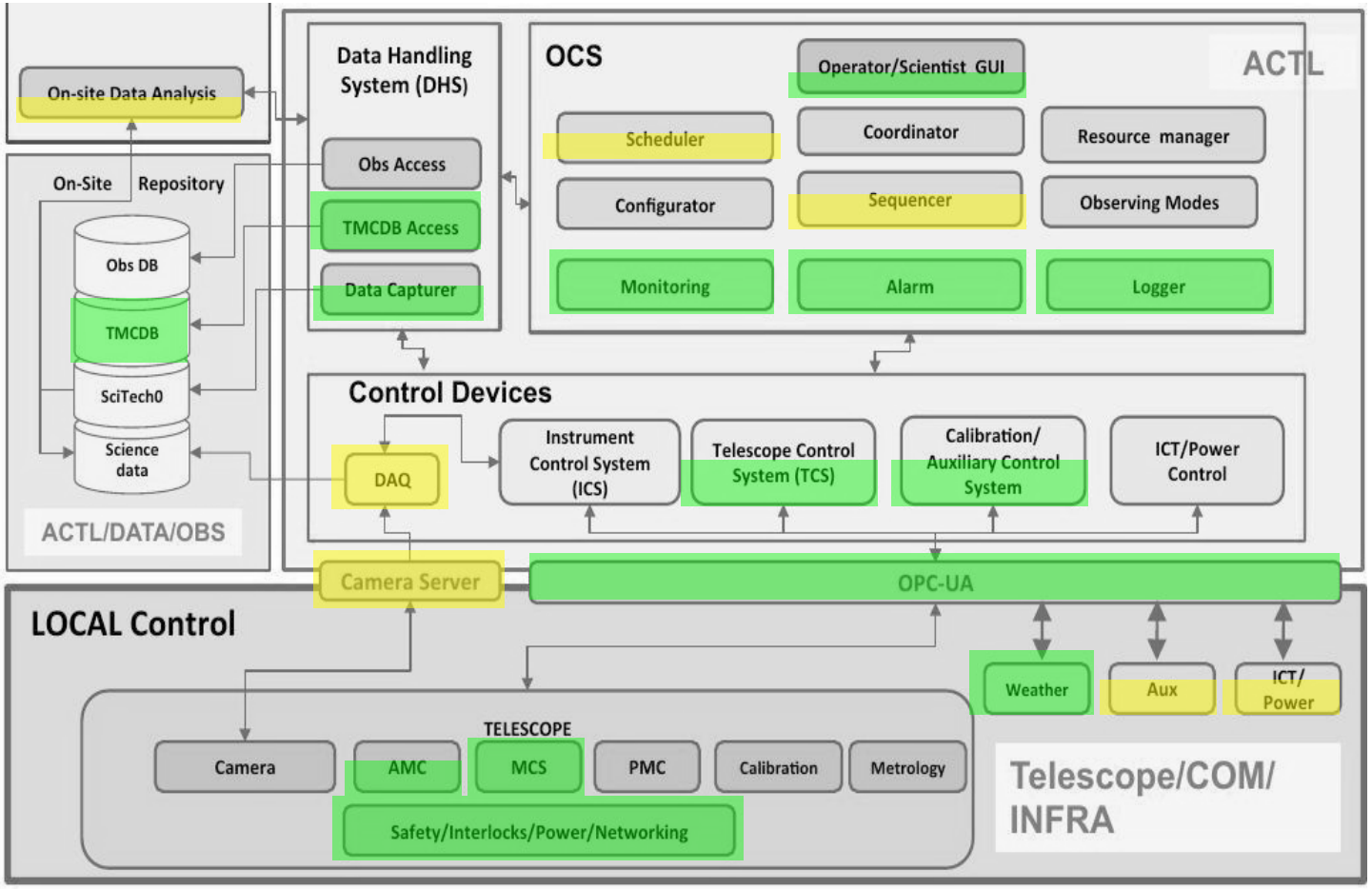
- AMC: ok
- CCD: ok (from MST)
- Drive: ok (from MST)
- Safety: in progress
- Digicam SC: in progress
- Digicam Data: in progress



SST-1M

Take advantages from diversity

Software

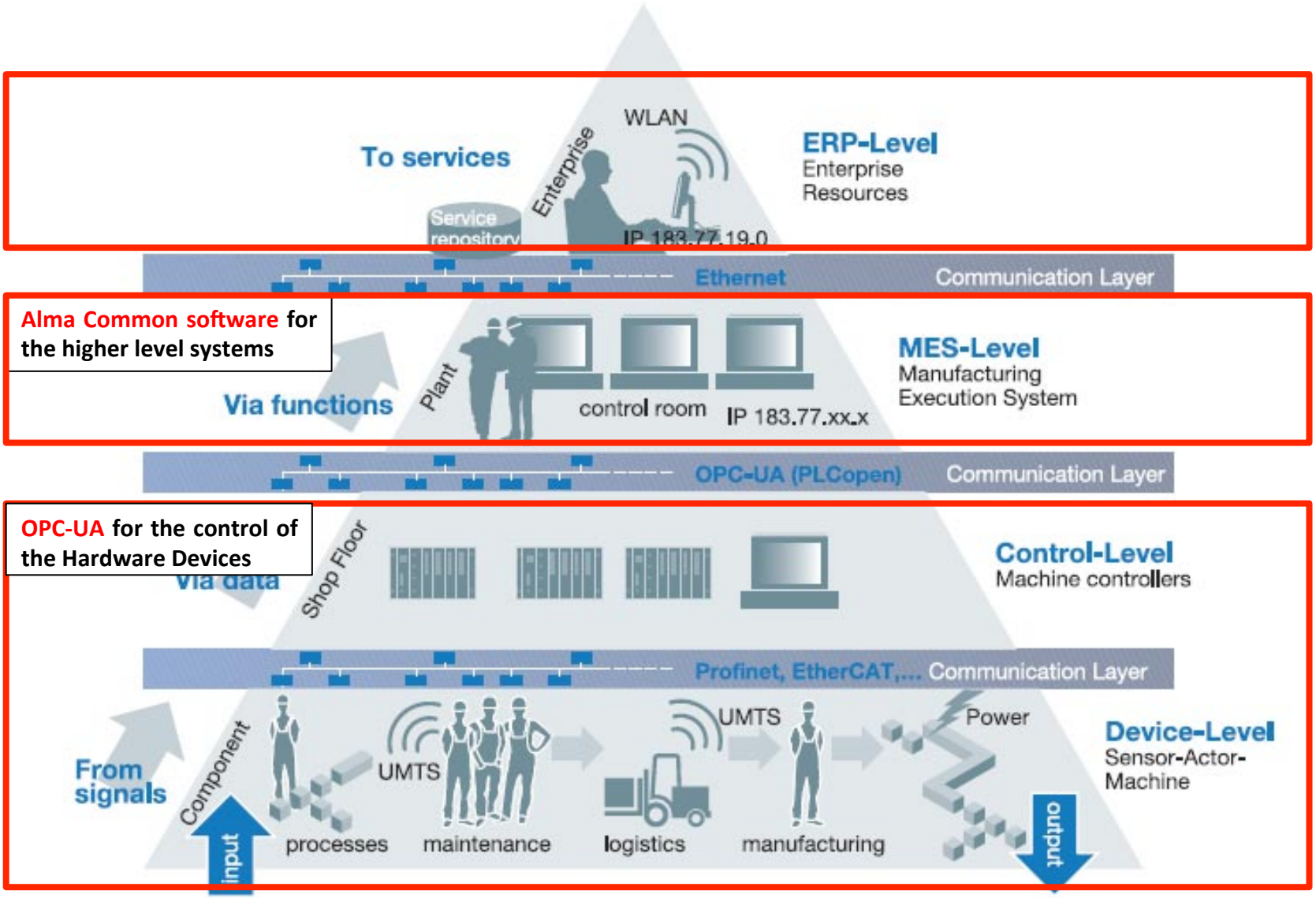


Deployed in Serra La Nave

Working in Lab

SST-2M

ACTL – SLOW and The Industrial Control System Pyramid



The discussion will proceed top-dow.

- The ACS Environment

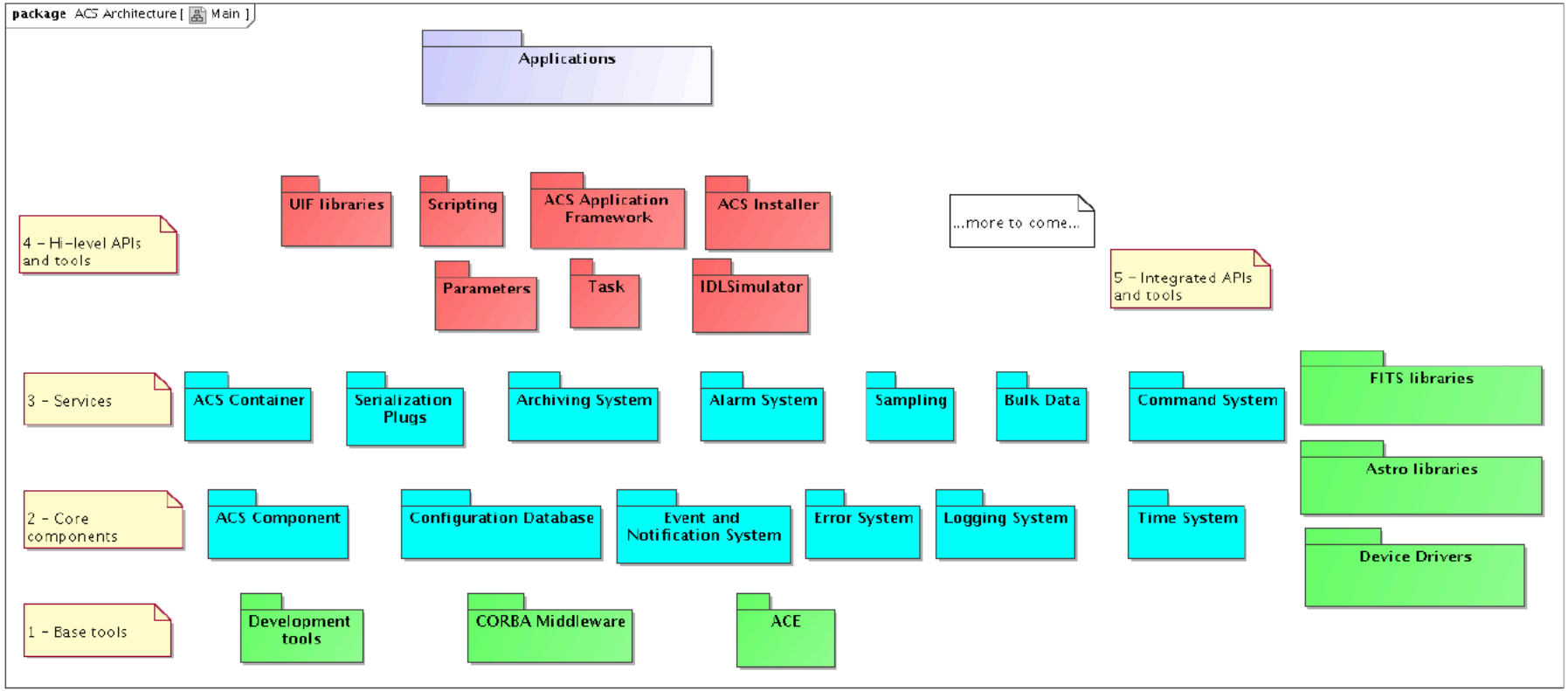


Figure 2.1: ACS Packages

ACS uses CORBA as middleware and provides:

- Application communication handling:
- Event Handling:
- Command:
- Logging:
- Persistent Store:
- Error/Alert Handling:

ACS Core Components

The components that are necessary for the development of any application are

- **ACS Component** Base interfaces and classes for Component part of the ACS Component Model. In particular C++ Distributed Objects, Properties and Characteristics are implemented in this package.
- **Configuration Database** Interfaces and basic implementation for the Configuration Database from where ACS Components retrieve their initial configuration
- **Event and Notification System** The Event and Notification System provides a generic mechanism to asynchronously pass information between data publishers and data subscribers, in a many to many relation scheme.
- **Error System** API for handling and logging runtime errors, tools for defining error conditions, tools for browsing and analyzing runtime errors.
- **Logging System** API for logging of data, actions and events. Transport of logs from the producer to the central archive. Tools for browsing logs.
- **Time System** Time and synchronization services.

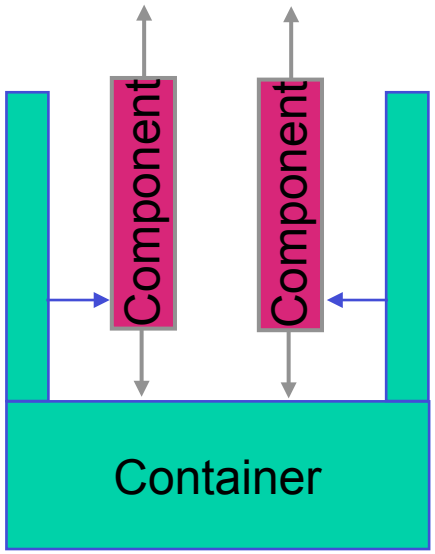
ACS Services

This services are not strictly necessary for the development of prototypes and test :

- **ACS Container** Design patterns, protocols and high level services for Component/ Container lifecycle management.
- **Serialization Plugs** This package provides a generic mechanism to serialize entity data between high level applications, typically written in Java.
- **Archiving System** API and services for archiving monitoring data and events from the run time system. Tools to browse, monitor and administer the flow of data toward the archive.
- **Command System** Tools for the definition of commands, API for runtime command syntax checking, API and tools for dynamic command invocation.
- **Alarm System** API and tools for configuration of hierarchical alarm conditions, API for requesting notification of alarms at the application level, tools for displaying and handling the list of active alarms.
- **Sampling** Low level engine and high level tools for fast data sampling (virtual oscilloscope).
- **Bulk Data** API and services for the transport of bulk science data (images or big data files) and continuous data streaming.

ACS Container-Component Model

- This is the primary instrument for achieving separation of functional from technical concerns.
- Every Component must implement the ComponentLifeCycle interface. This interface foresees the following basic lifecycle operations:
- initialize – called to give the component time to initialize itself, e.g. retrieve connections, read configuration parameters, etc
- execute – called after initialize() to tell the component that it has to be ready to accept incoming functional calls any time
- cleanup – the component should release resources in an orderly manner, because shutdown is imminent
- aboutToAbort – the component will be forcibly removed due to some error condition.
- The Container passes to each Component a ContainerServices object. At that point, the Component can assume that all infrastructural services it may need have been properly set up by the Container and are available via the ContainerServices object.



Characteristic Components

- Characteristic Components are a subclass of Components tailored to the implementation of objects that describe collections of numerical (typically physical) quantities. They have been designed in particular to represent Control System objects with monitor and control points or objects with state and configurable parameters?Characteristic
- At control system level, Characteristic Component is the base class used for the representation of any physical (a temperature sensor, a motor) or logical device in the control system.
- Higher level applications can use Characteristic Components to implement any Component that has configurable values representing numerical quantities.

Properties

- Each Characteristic Component has 0..n Properties that are monitored and controlled, for example status, position, velocity and electric current.

- Properties can be read only or read/write.
- Properties can represent values using a limited set of basic data types:
- *long*, *longLong* and *uLongLong* for integers
- *double* for floating point numbers
- *string* for strings.
- *enum* for enumerations like states. This includes a boolean TRUE/FALSE enumeration.
- *sequence<scalar property>* of one of the previously defined scalar property types. For example a *sequence<long>* allows manipulating a group of properties of type long. Each item in the list can be assigned to a long property object and manipulated (reading characteristics and value) independently from the others.

Characteristics

- Static data associated with a Characteristic Component or with a Property, including metadata such as:
- *name, description, version* and *dimensions*
- other data such as *units, range* or *resolution.* Each Characteristic Component or each Property has 0..n Characteristics
- Characteristic Components and Properties: Name, Description, Version etc.
- Readonly and Read/Write Properties: default values, range, units, format, resolution

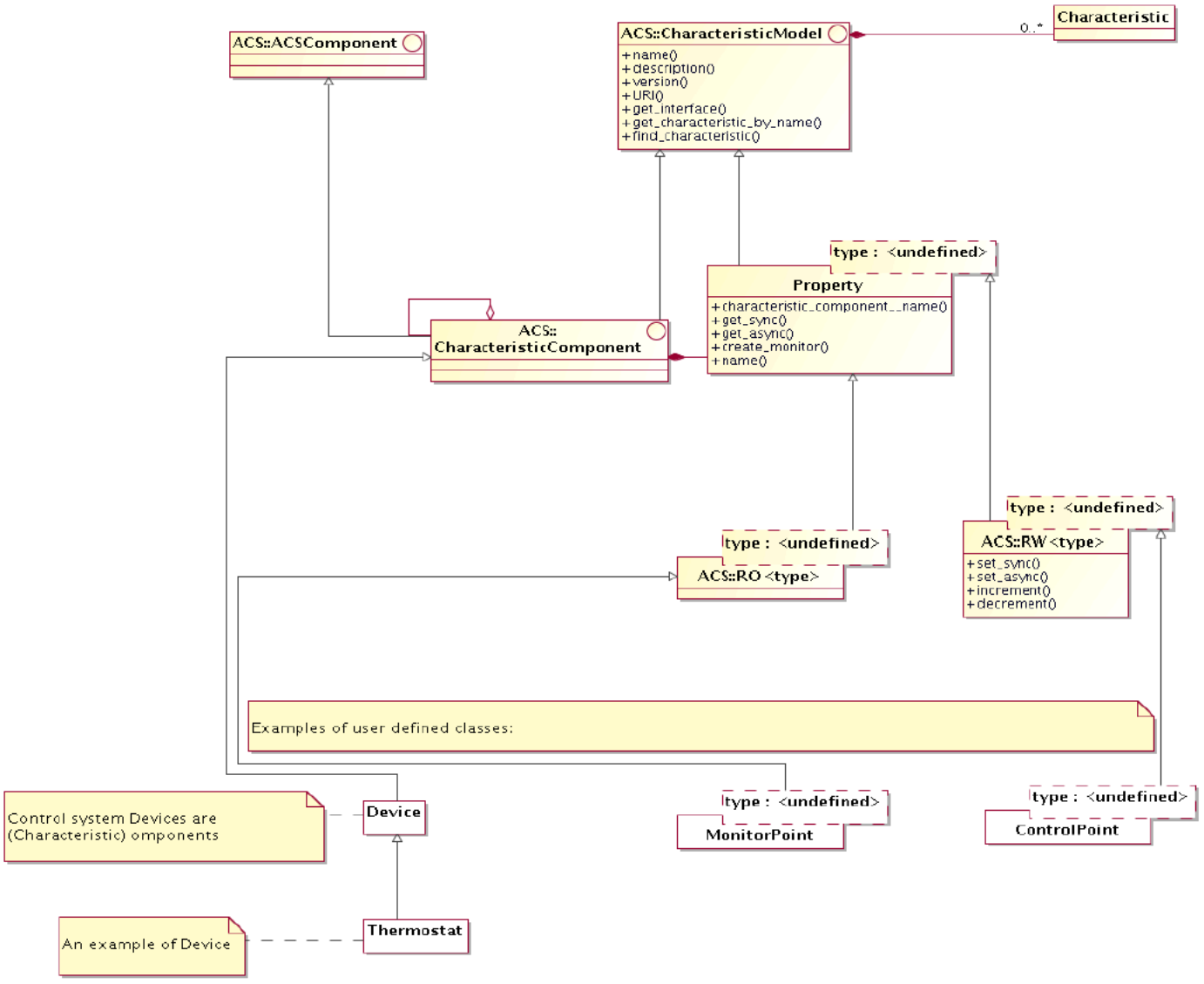


Figure 3.1: Characteristic Component - Property - Characteristic class diagram

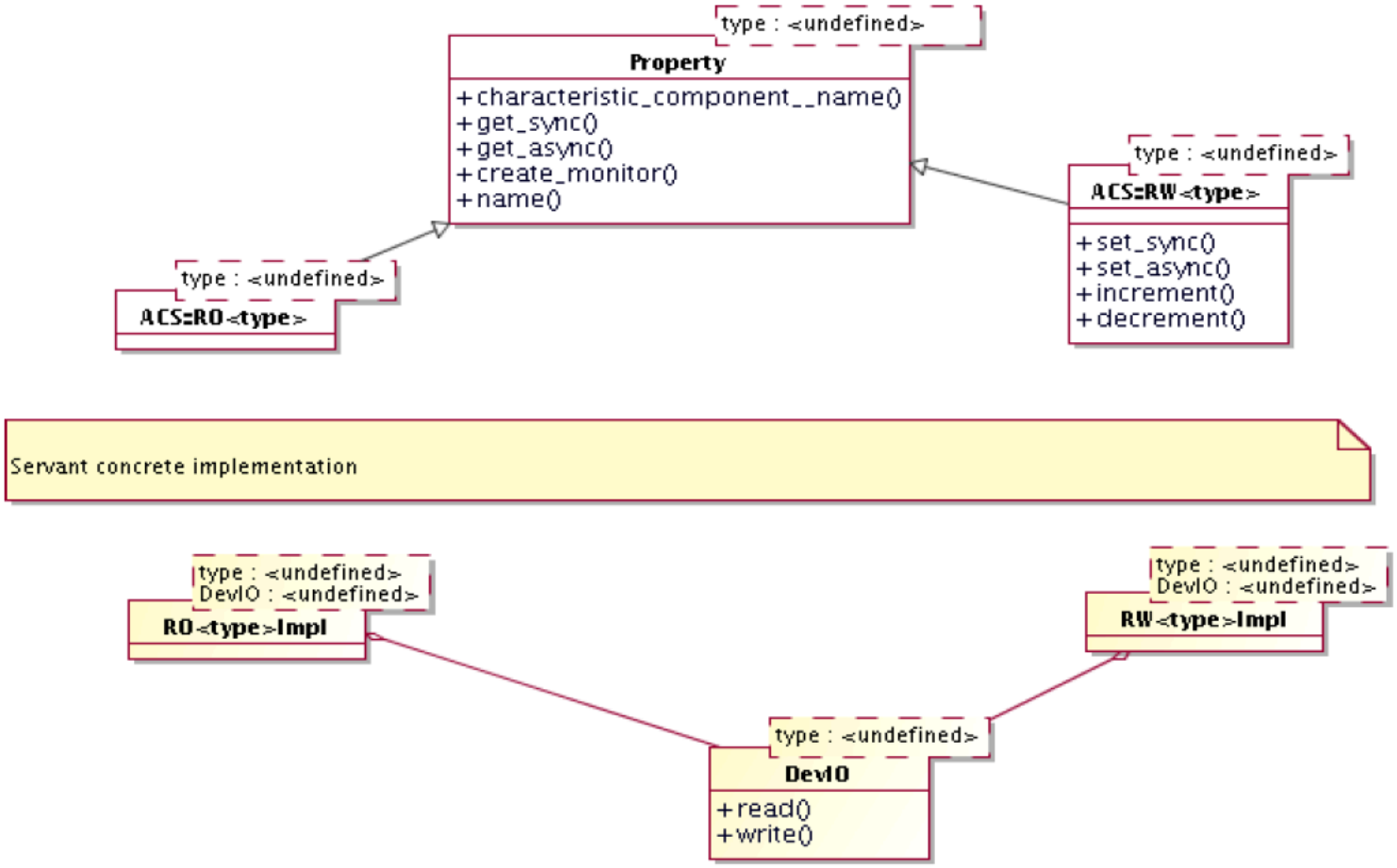
Properties & Hardware

- The classes implementing the Property interfaces are responsible for the actual interfacing with the hardware or, more in general, to retrieve/ calculate the value for the numerical entities. In order to decouple as much as possible the implementation of Property classes and the access to different kinds of data sources, ACS provides a parametrized Property implementation based on the DevIO parameter,

- A DevIO implementation is responsible only for reading/writing the Property's value from a specific device (memory location, OPC-UA nodes in CTA).

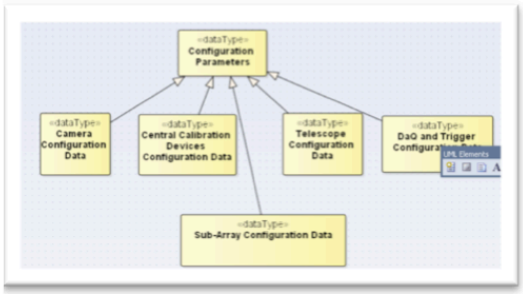
- The configuration parameters for all Characteristic Components, i.e. the initial values for Properties control values and all Characteristics for Properties, are persistently stored in the Configuration Database.

DevIO


type : <undefined>
Property
+characteristic_component__name()
+get_sync()
+get_async()
+create_monitor()
+name()
type : <undefined>
ACS::RW<type>
+set_sync()
+set_async()
+increment()
+decrement()
type : <undefined>
ACS::RO<type>
Servant concrete implementation
type : <undefined>
DevIO : <undefined>
RO<type>Impl
type : <undefined>
DevIO : <undefined>
RW<type>Impl
type : <undefined>
DevIO
+read()
+write()

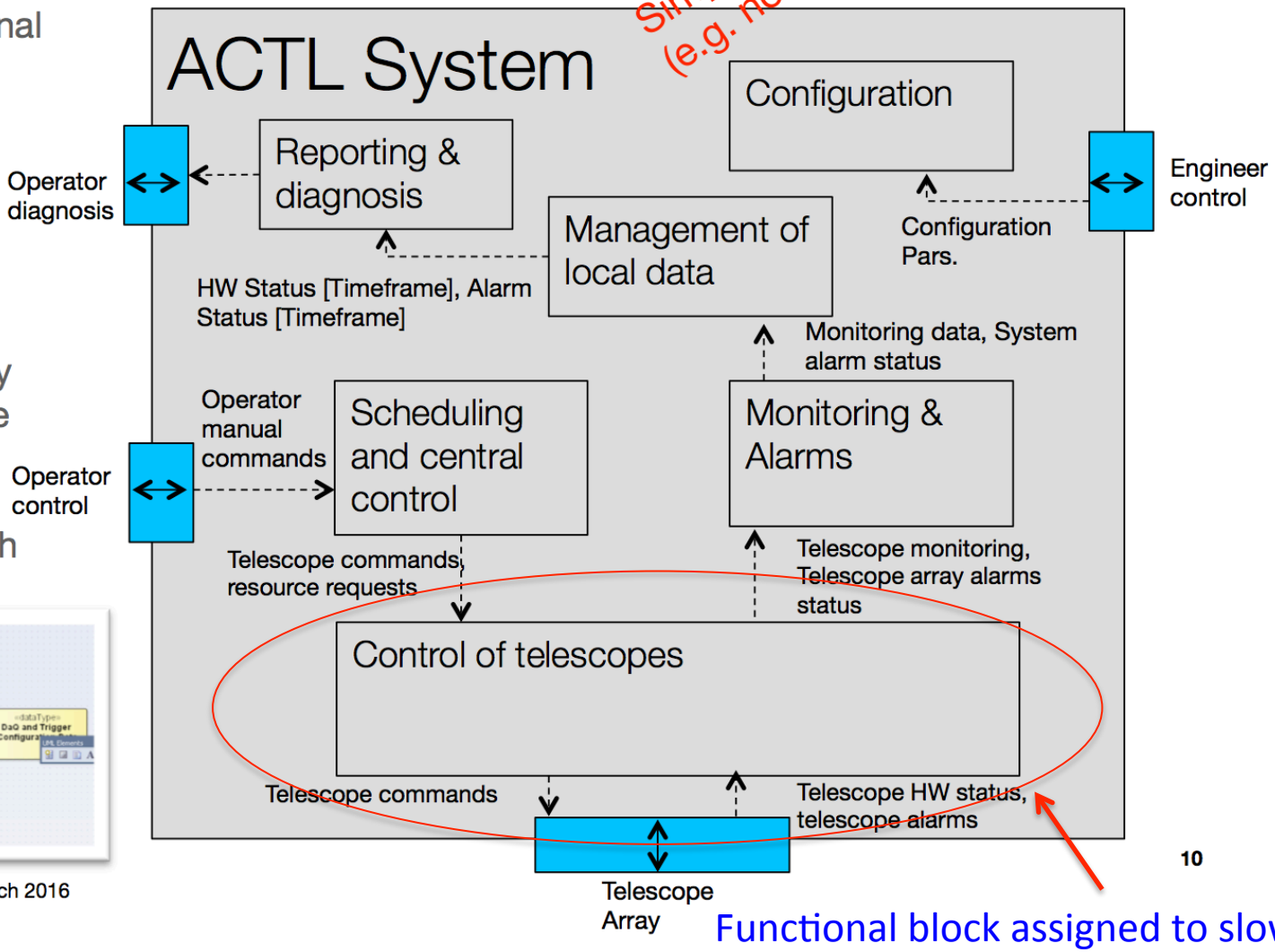
Functional View – ACTL as a white box

- Information flow of 'abstract', functional data elements
- System decomposed in functions
- Functions have structure and granularity
- Function hierarchy allow to create the WBS
- Functional data associated to each flow



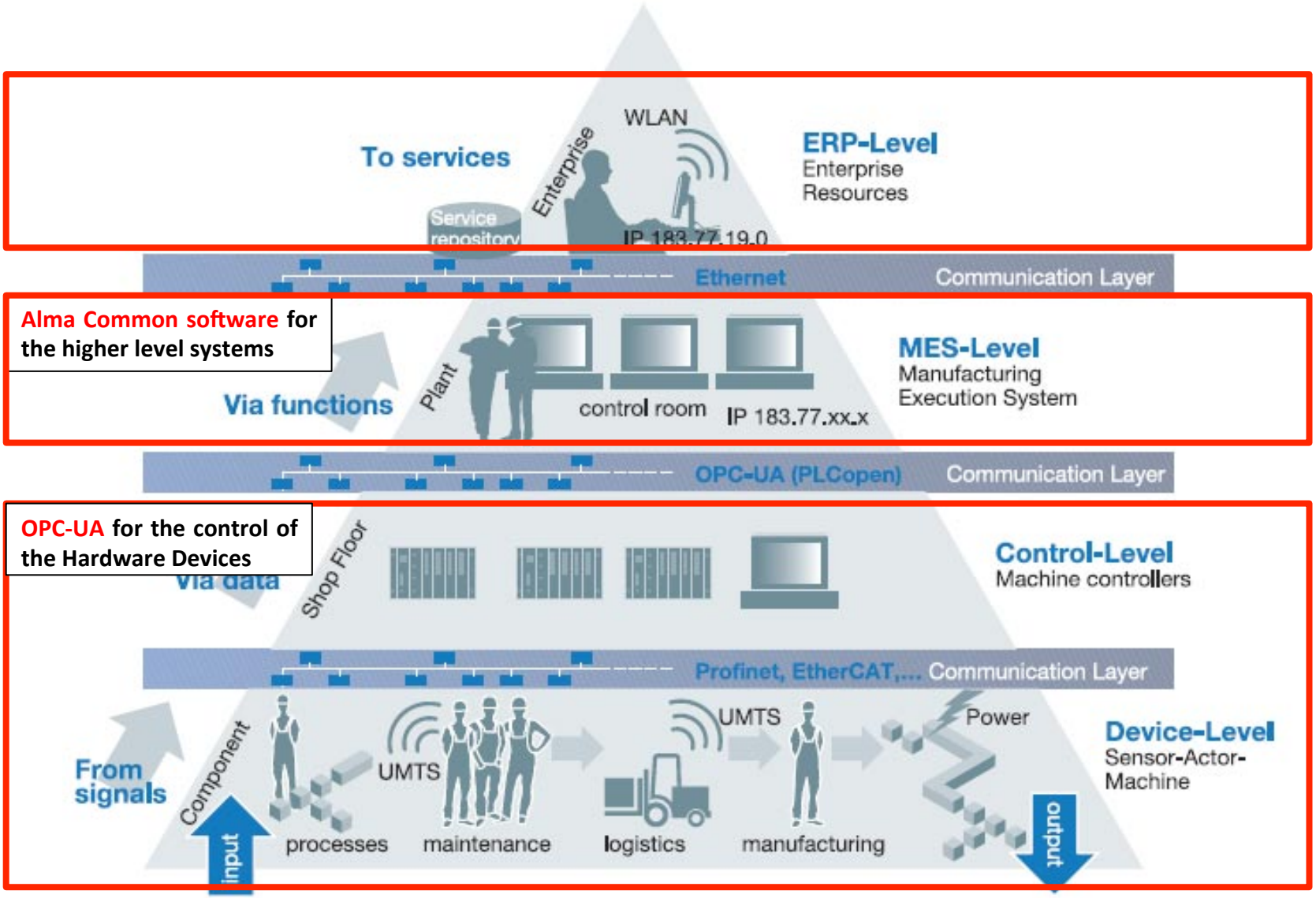
Igor Oya, MST meeting, March 2016

Simple example (e.g. no DAQ)

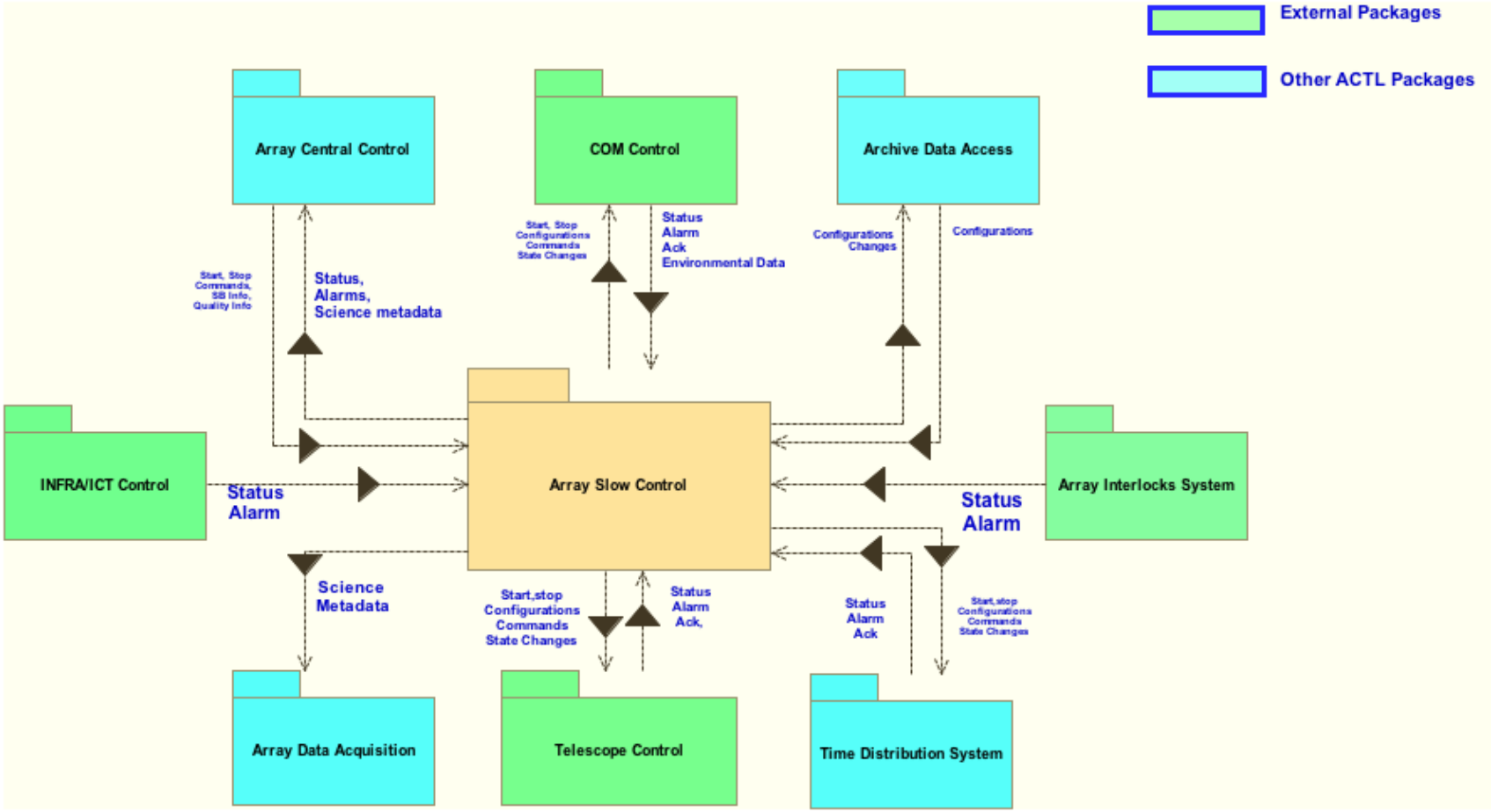


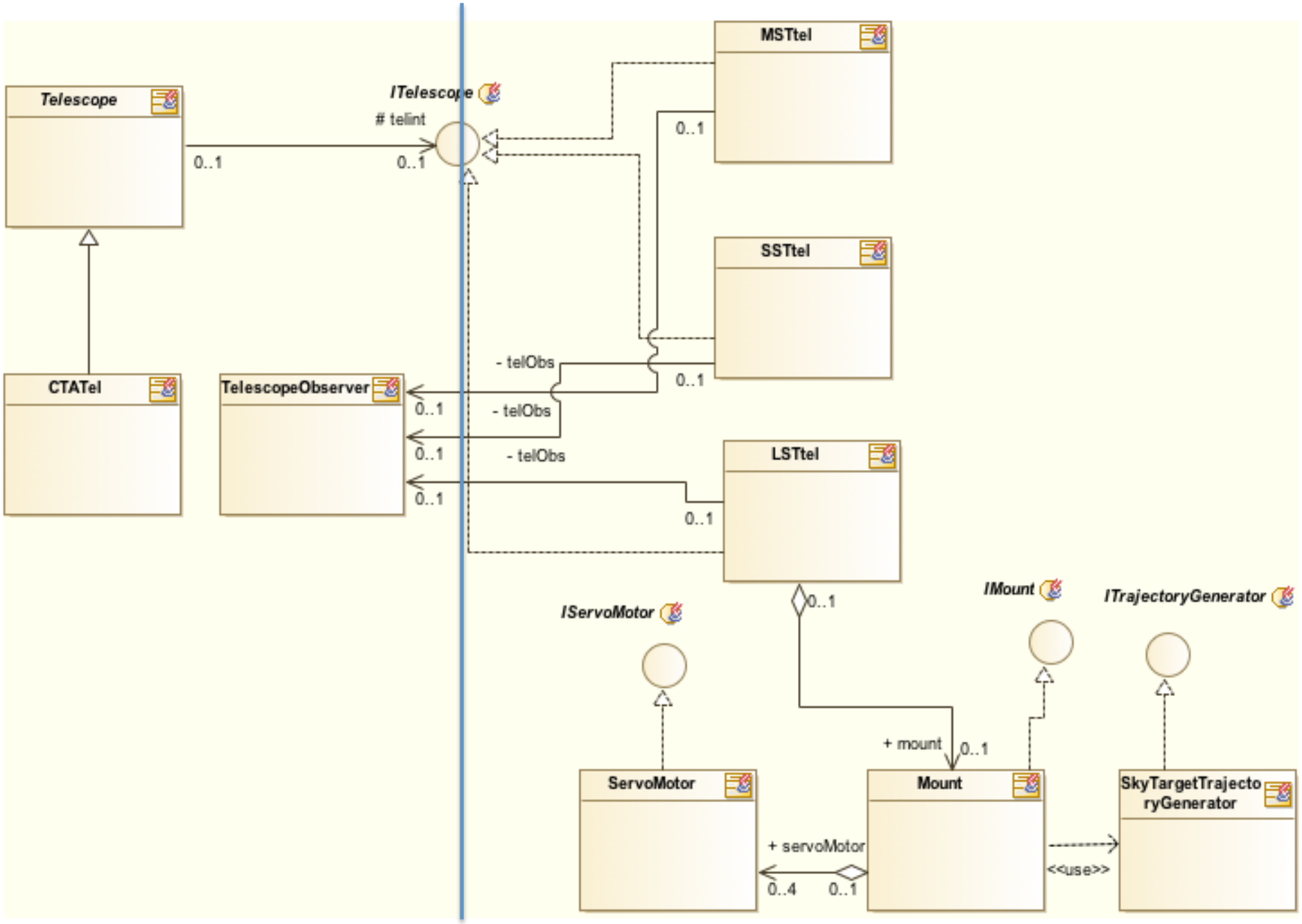
Functional block assigned to slow

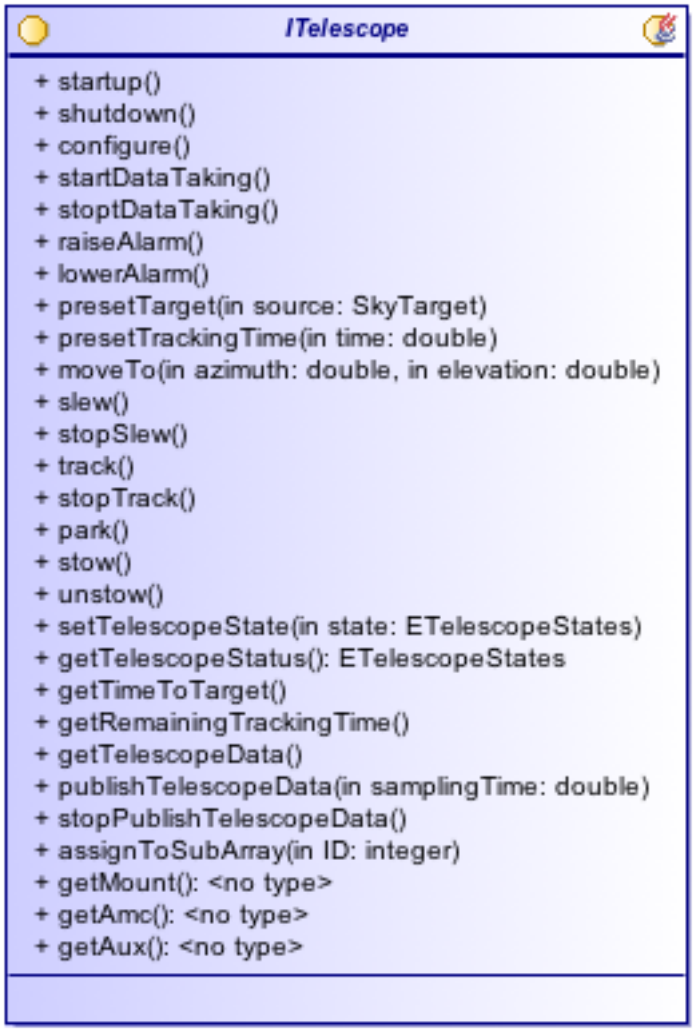
ACTL – SLOW and The Industrial Control System Pyramid

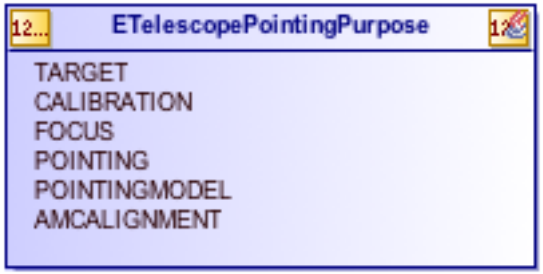


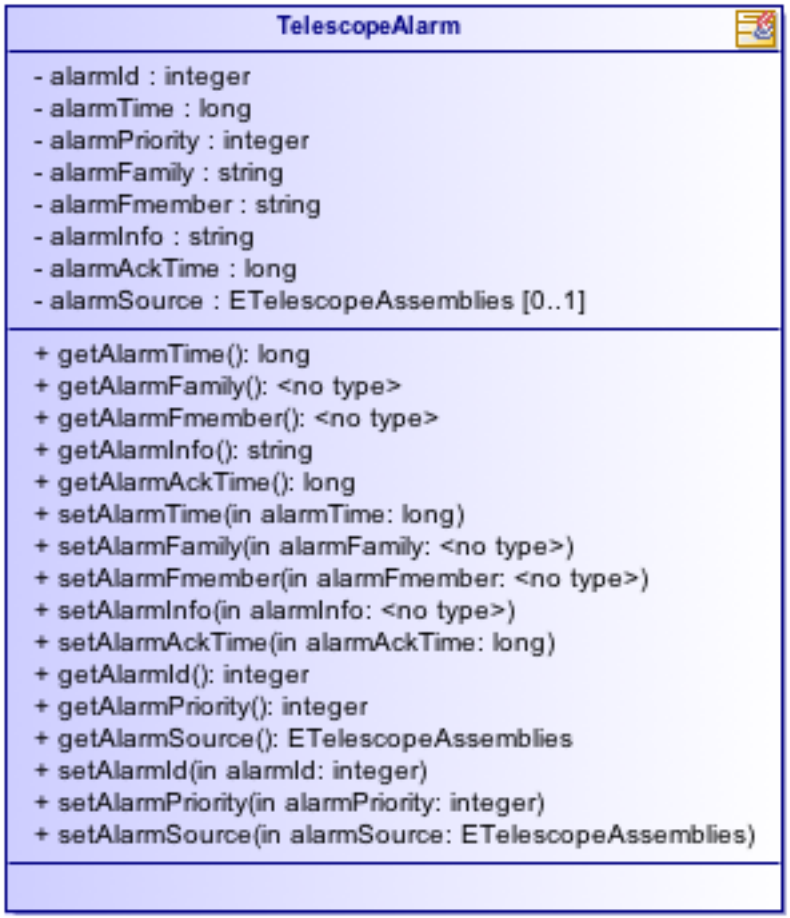
ACTL-SLOW Context




Telescope
ITelescope
telint
0..1
0..1
MSTtel
0..1
SSTtel
0..1
CTATel
TelescopeObserver
- telObs
- telObs
- telObs
0..1
0..1
0..1
LSTtel
0..1
0..1
IServoMotor
IMount
ITrajectoryGenerator
+ mount
0..1
ServoMotor
Mount
SkyTargetTrajectoryGenerator
+ servoMotor
0..4
0..1
<<use>>

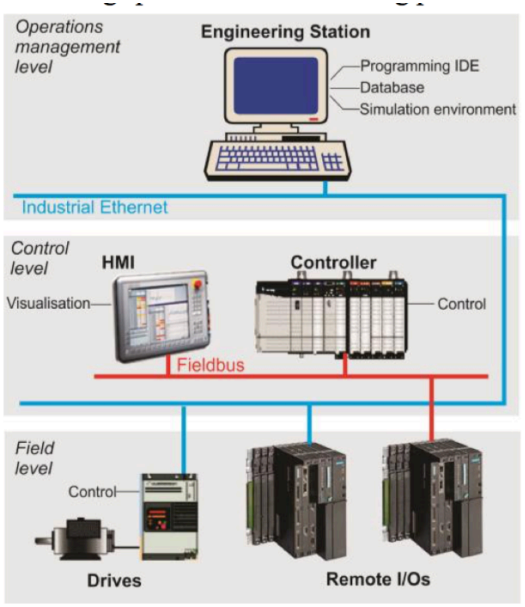

ITelescope
+ startup()
+ shutdown()
+ configure()
+ startDataTaking()
+ stoptDataTaking()
+ raiseAlarm()
+ lowerAlarm()
+ presetTarget(in source: SkyTarget)
+ presetTrackingTime(in time: double)
+ moveTo(in azimuth: double, in elevation: double)
+ slew()
+ stopSlew()
+ track()
+ stopTrack()
+ park()
+ stow()
+ unstow()
+ setTelescopeState(in state: ETelescopeStates)
+ getTelescopeStatus(): ETelescopeStates
+ getTimeToTarget()
+ getRemainingTrackingTime()
+ getTelescopeData()
+ publishTelescopeData(in samplingTime: double)
+ stopPublishTelescopeData()
+ assignToSubArray(in ID: integer)
+ getMount(): <no type>
+ getAmc(): <no type>
+ getAux(): <no type>

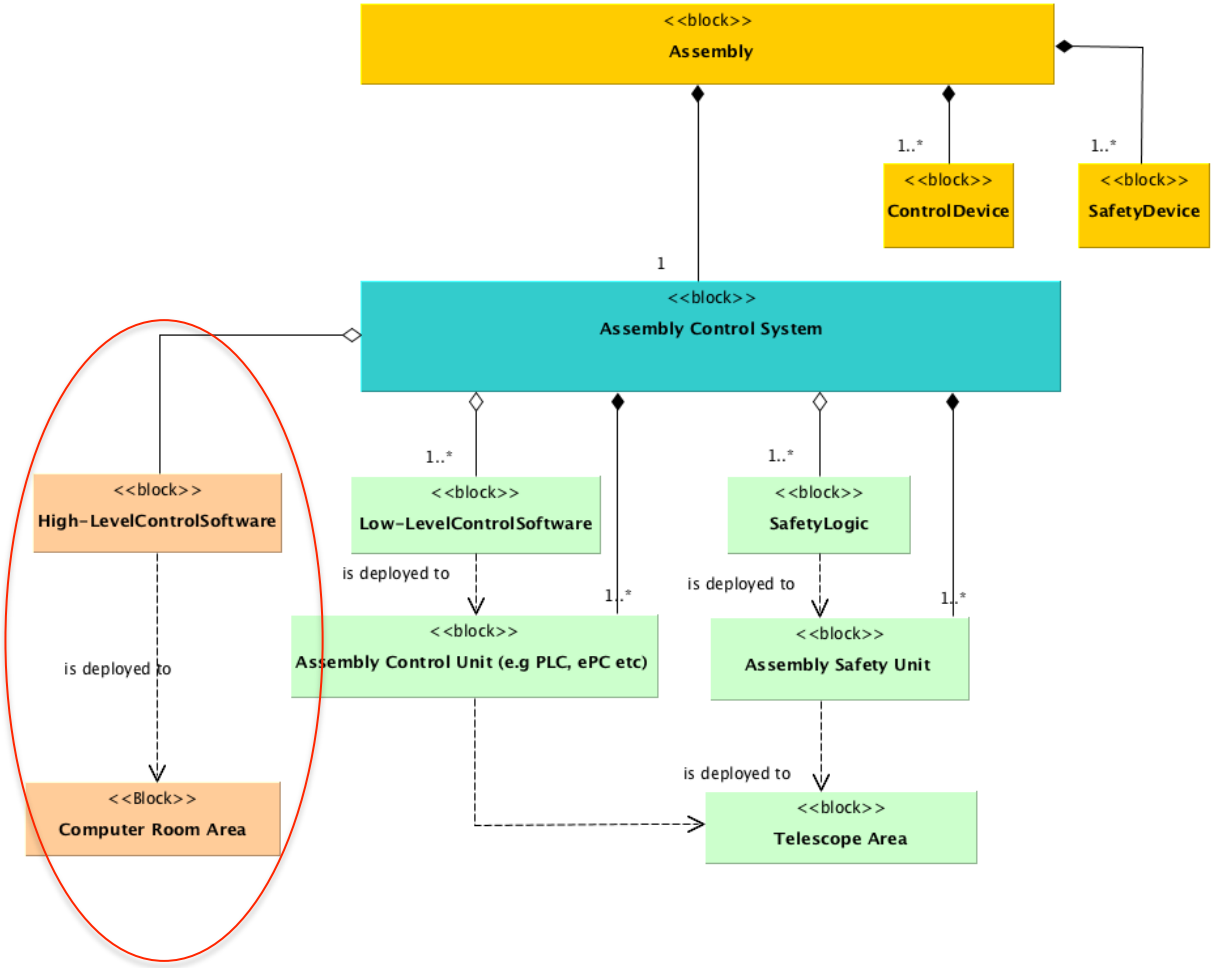

ETelescopePointingPurpose
TARGET
CALIBRATION
FOCUS
POINTING
POINTINGMODEL
AMCALIGNMENT


TelescopeAlarm
- alarmId : integer
- alarmTime : long
- alarmPriority : integer
- alarmFamily : string
- alarmFmember : string
- alarmInfo : string
- alarmAckTime : long
- alarmSource : ETelescopeAssemblies [0..1]
+ getAlarmTime(): long
+ getAlarmFamily(): <no type>
+ getAlarmFmember(): <no type>
+ getAlarmInfo(): string
+ getAlarmAckTime(): long
+ setAlarmTime(in alarmTime: long)
+ setAlarmFamily(in alarmFamily: <no type>)
+ setAlarmFmember(in alarmFmember: <no type>)
+ setAlarmInfo(in alarmInfo: <no type>)
+ setAlarmAckTime(in alarmAckTime: long)
+ getAlarmId(): integer
+ getAlarmPriority(): integer
+ getAlarmSource(): ETelescopeAssemblies
+ setAlarmId(in alarmId: integer)
+ setAlarmPriority(in alarmPriority: integer)
+ setAlarmSource(in alarmSource: ETelescopeAssemblies)

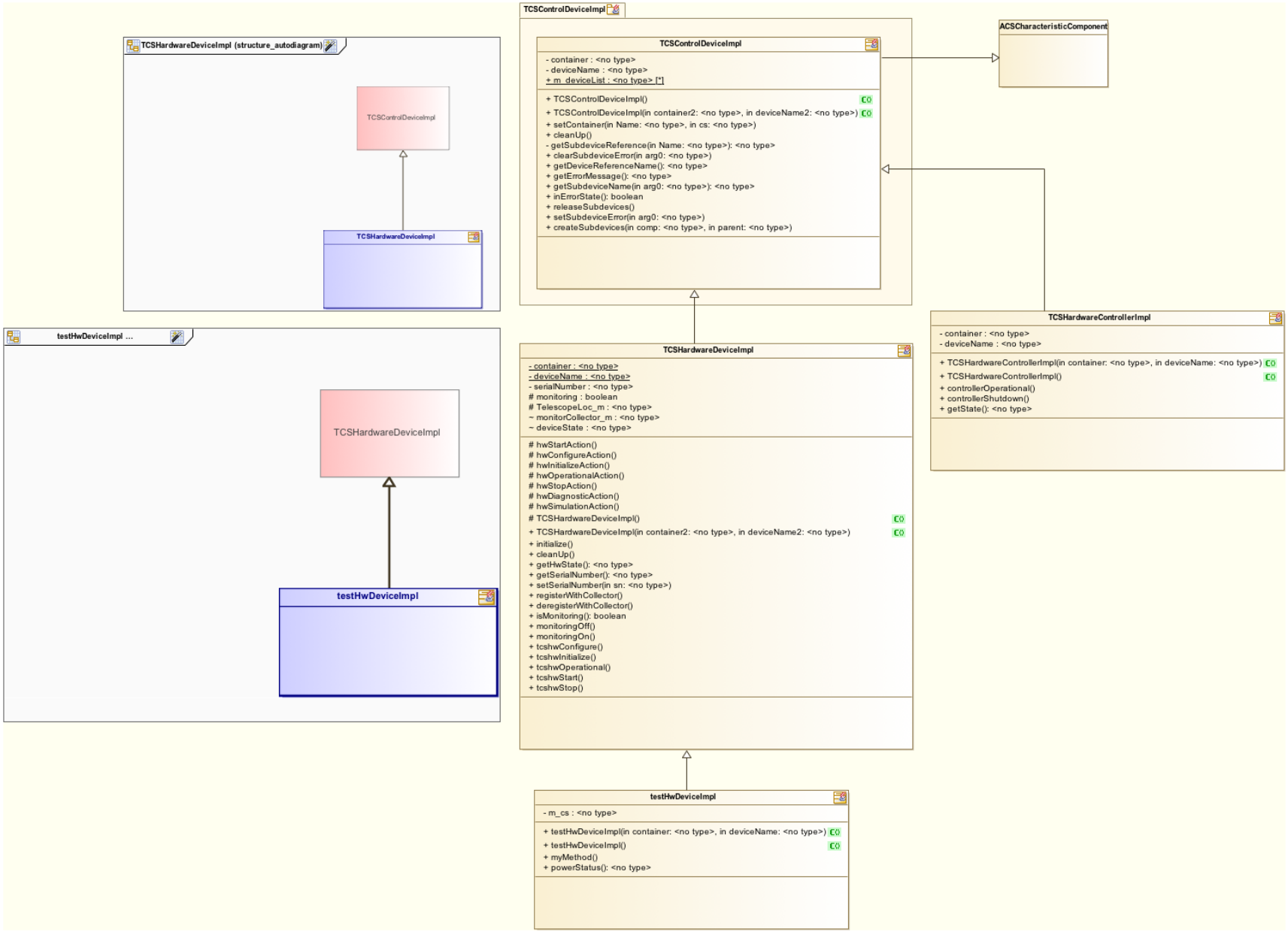

ETelescopeStates
OFFLINE
SAFE
OPERATIONAL
MAINTENANCE
FAULT

Assembly Control system

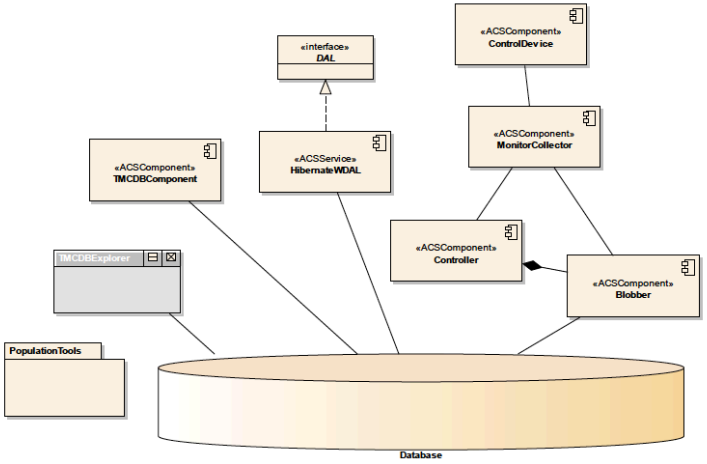


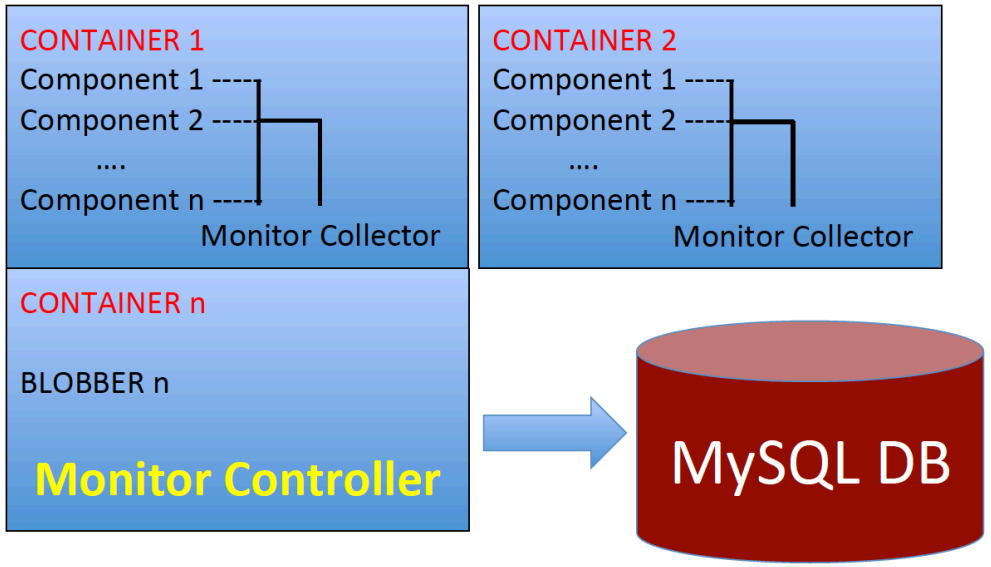


Hardware Devices



Hardware Device and TMCDB



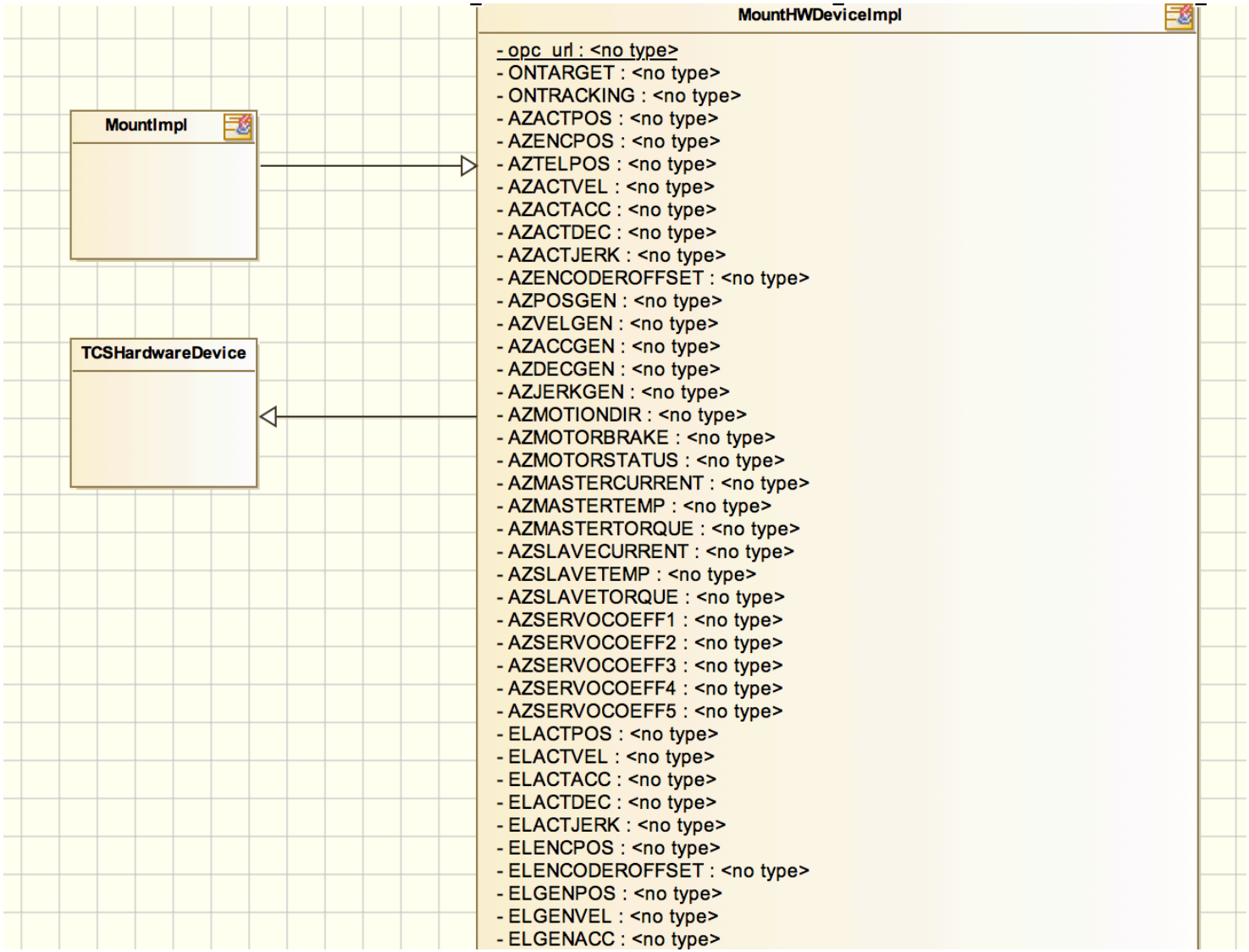


Just an Example: You can replace MySQL with any other DB engine.

The Hardware device already provide the basic methods to store monitoring data in the Monitoring database

For prototyping and test you can just activate a monitor to each desired property and store tables on files. A few python lines would be enough in this case.

Real Example: The Mount



ACTL SLOW DATA Model and Hardware device ICD

- An Hardware Devices is an ACS component, that implements the ICD of that device. Its just that: Monitor Points (MPs) and Control/Set Points (CPs) for each entry in the ICD.

- Each Hardware device uses OPC-UA Node address to be able to communicate with a particular MP and extract its value, or to command a certain CP to a new value.

- CONTROL devices also include whatever lifecycle, transformation of data, and specials programming that the ICD specifies.

ACTL-SLOW Data Model

The data model for a given hardware device contains important information and serves the following practical purposes:

- Provide a conceptual description of the objects in the system and their relationships.
- Provide a blueprint for creating the TMCDB database structure.
- Guide implementation of code units that access the database.
- Serve as input for automatic generation of database schema and data access code.
- Define the contents of the ICDs

Data Model and OPC-UA

Data Model for each hardware device in the System is very important because parts of their OPC UA servers may be automatically generated.

In this way hand-written custom code is only necessary for providing business logic between the generated parts and software handling the specific device type (e.g. a hardware access library or protocol implementation for an physical device).

Higher level business logic can be implemented in the ACS hardware device side.

Data Model and OPC-UA

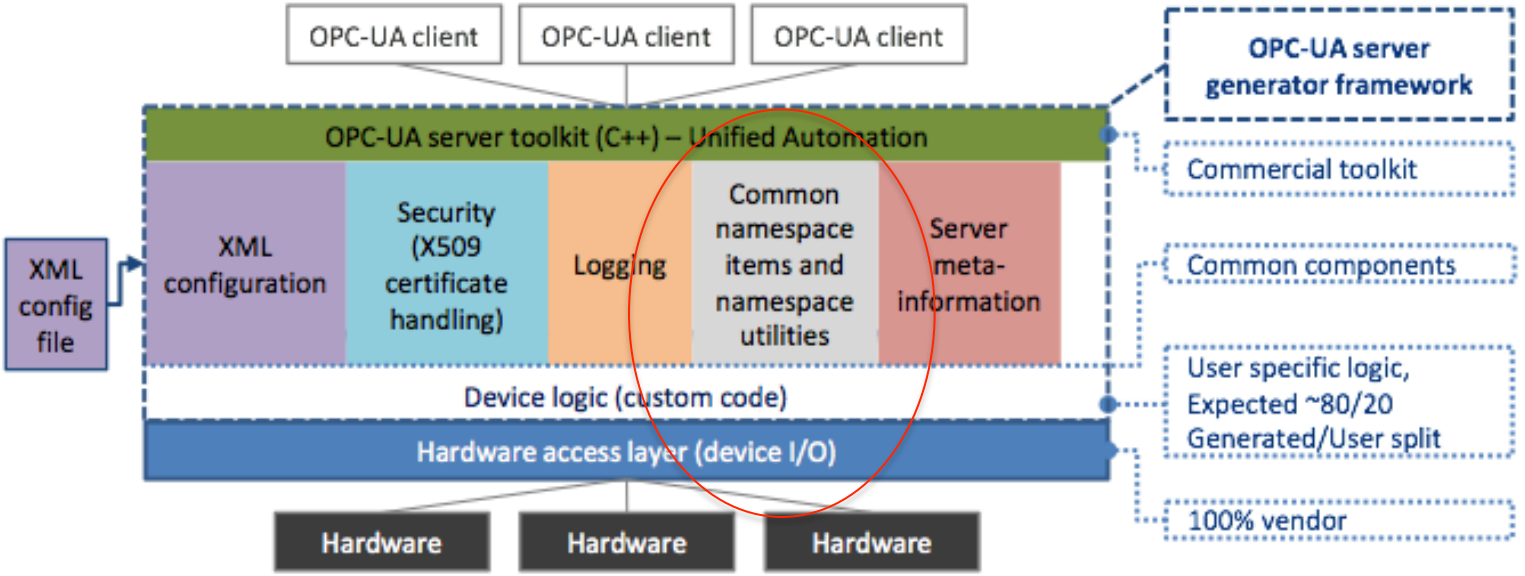


Figure 1. Overview of the generic server framework components.

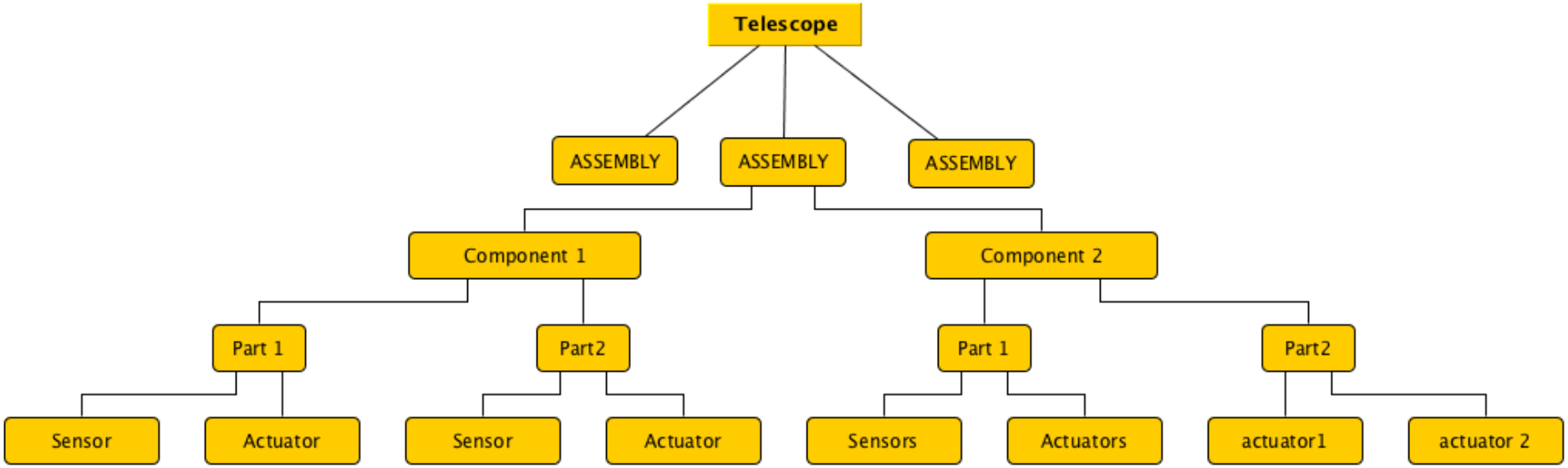
Development can starts with creation the data model (or design) , in XML, or other format , describing an object-oriented information model of the target system or device.

Here we propose a path to define the main data items or entities of the hardware device for the OPC-UA server and for its mapping in the ACS High level components. These information will be then used by ACTL-Slow to design the ACTL repository.

Uniform CPs & MPs definition

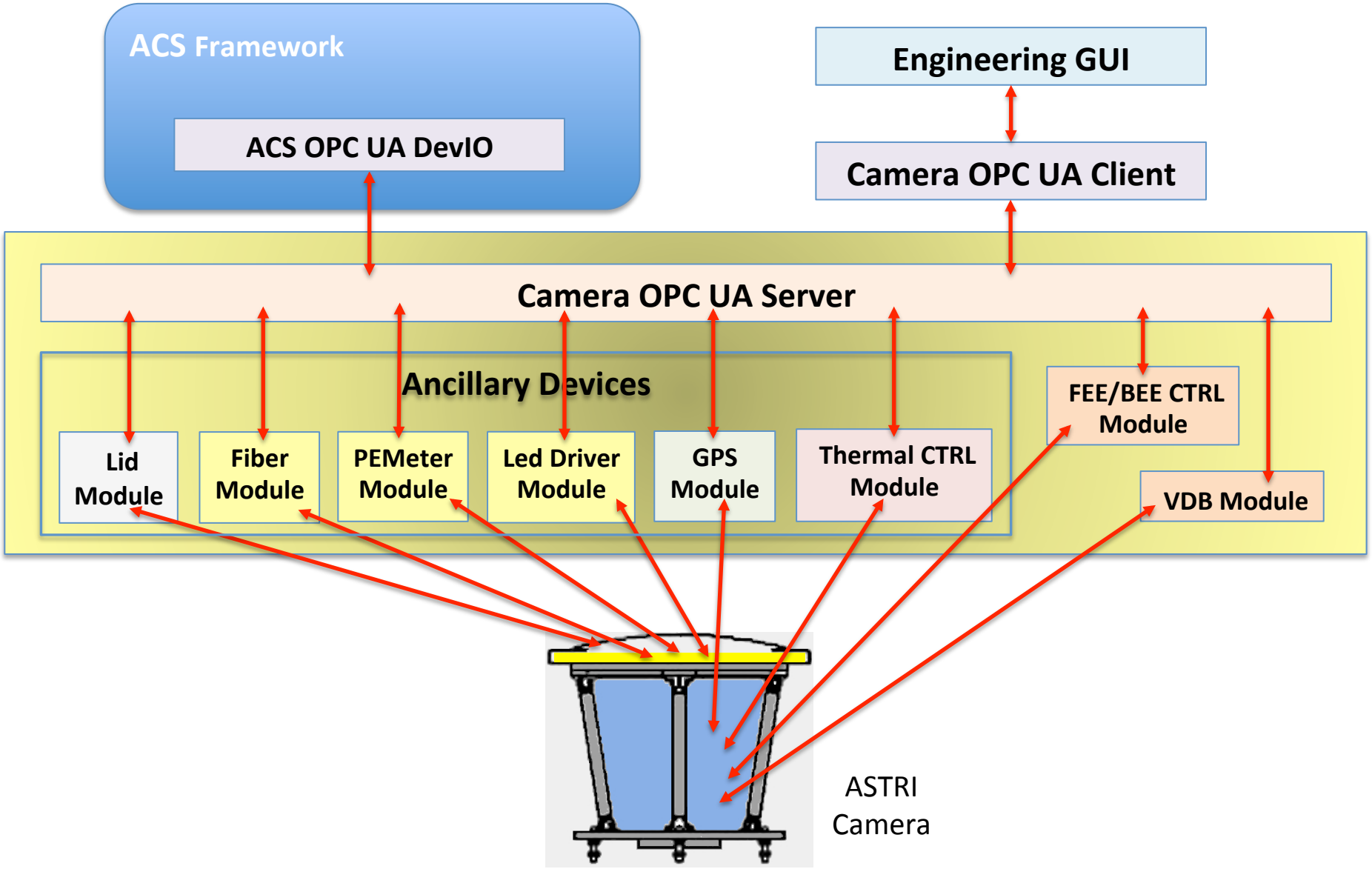
- In order to define an uniform naming convention for the control and monitor points of the telescope, the first step is to use the telescope Product tree document to extract the list of the Assemblies, Component and parts.
- From this we can extract the list of common devices (eg. Motors, I/O) and the list of the measurement, command, safety and configuration data items (in an abstract way).
- In conclusion we need to define the Data Model associated with all Hardware device of the Telescope Array.

1-The Hardware device Hierarchy



We would ask Telescope teams to provide their hardware device hierarchy

Example of a hardware device internal hierarchy



Example of a Hardware device Hierarchy with the variables associated to each component and parts

MOUNT TREE COMPONENTS – HARDWARE MONITORING

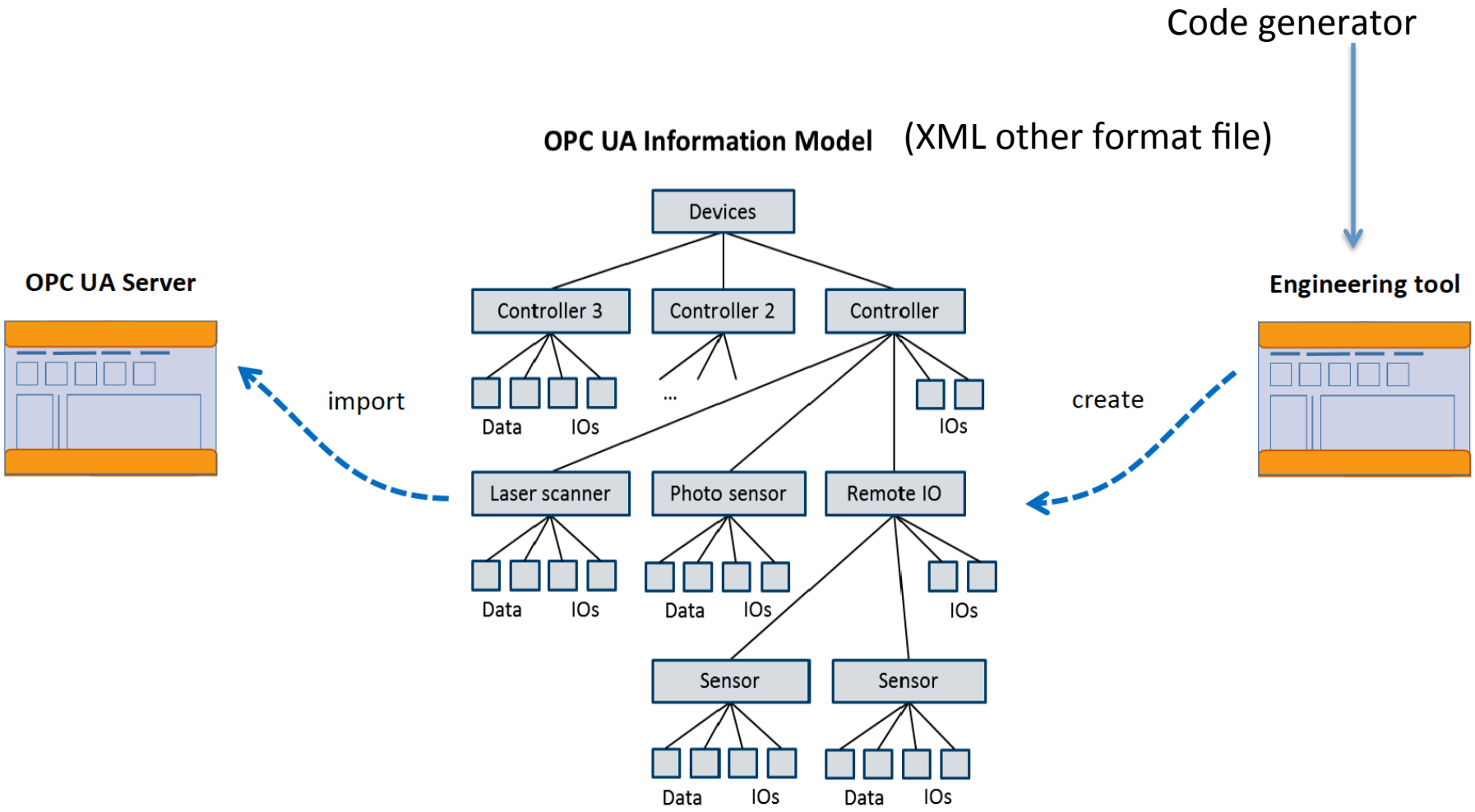
- MOUNT
  - TCU (PLC)
    - DIGITAL I/O (switches)
      - ✓ Lyre - Status (on/off)
      - ✓ AZ CW proximity switch - status (on/off)
      - ✓ AZ CCW proximity switch - status (on/off)
      - ✓ AZ park position proximity switch – status (on/off)
      - ✓ EL horizon proximity switch - status (on/off)
      - ✓ EL zenith proximity switch - status (on/off)
      - ✓ EL park position proximity switch - status (on/off)
    - AZ ENCODERS (Incremental strip encoder – AZ actual position)
      - ✓ ENCODER1 - Status, Latch, Count
      - ✓ ENCODER2 - Status, Latch, Count
      - ✓ ENCODER3 - Status, Latch, Count
      - ✓ ENCODER4 - Status, Latch, Count
    - EL ENCODER (Absolute - EL actual position)
      - ✓ Status, Latch, Count
    - SERVO
      - ✓ AZ MOTOR1
        - _ Temperature, Current, Torque, Brake.
        - _ acceleration, deceleration, jerk, direction
      - ✓ AZ MOTOR2
        - _ Temperature, Current, Torque.
        - _ acceleration, deceleration, jerk, direction
      - ✓ EL MOTOR
        - _ Temperature, Current, Torque, Brake.
        - _ acceleration, deceleration, jerk, direction
    - SERVO DRIVES
      - ✓ AZ SERVO1 – status (e.g. enabled, disabled, operative)
      - ✓ AZ SERVO2 - status (e.g. enabled, disabled, operative)
      - ✓ EL SERVO – status (e.g. enabled, disabled, operative)
    - FIELDBUS
      - ✓ EtherCAT Master – status and NetID
      - ✓ EtherCAT Slaves – status and address

      (e.g. Digital I/O modules, encoder interfaces, servo drive interfaces)
  - THCU (PLC)
    - SAFETY DIGITAL I/O (Interlocks)
      - ✓ AZ CW Operational limit switch – status
      - ✓ AZ CW Emergency limit switch – status
      - ✓ AZ CCW Operational limit switch – status
      - ✓ AZ CCW Emergency limit switch – status
      - ✓ AZ park position limit switch - status
      - ✓ AZ stow pin insertion limit switch – status
      - ✓ AZ stow pin extraction limit switch – status
      - ✓ EL horizon Operational limit switch – status
      - ✓ EL horizon Emergency limit switch – status
      - ✓ EL zenith Operational limit switch – status
      - ✓ EL zenith Emergency limit switch – status
      - ✓ EL park position limit switch - status
      - ✓ EL stow pin insertion limit switch – status
      - ✓ EL stow pin extraction limit switch – status
      - ✓ Door base emergency switch – status
      - ✓ Door cabinets emergency switch – status
      - ✓ Emergency Stop1 mushroom – status
      - ✓ Emergency Stop2 mushroom – status
      - ✓ Emergency Stop3 mushroom – status
      - ✓ AZ motor safe Stop (24 V to the drive) - status
      - ✓ EL motor safe Stop (24Vto the drive) - status
    - DIGITAL I/O (Status of the Telescope devices)
      - ✓ Motors high voltage on/off – status
      - ✓ Camera on/off – status
      - ✓ Data Logger on/off – status
      - ✓ Mirror1 on/off – status
      - ✓ Mirror2 on/off – status
      - ✓ Pointing Monitor Camera on/off – status
      - ✓ Camera Thermal Control on/off – status
      - ✓ Main Disconnector on/off – status
      - ✓ Sky Quality Meter on/off – status
      - ✓ UVSiPM on/off – status
      - ✓ UVScope on/off – status
      - ✓ Active Mirror Control Unit on/off – status
      - ✓ Telescope Control Unit on/off – status
      - ✓ SQM-PMC Heather on/off – status
    - ANALOG I/O
      - ✓ HPC Cabinet Temperature
      - ✓ LPC Cabinet Temperature
    - POWER MEASUREMENT

      (This terminal enables the measurement of all relevant electrical data of the supply network.)
      - ✓ HIGH-VOLTAGE PHASE 1 – current, voltage, power, energy, cosphi, frequency
      - ✓ HIGH-VOLTAGE PHASE 2 – current, voltage, power, energy, cosphi, frequency
      - ✓ HIGH-VOLTAGE PHASE 3 – current, voltage, power, energy, cosphi, frequency
    - FIELDBUS
      - ✓ EtherCAT Master – status and NetID
      - ✓ EtherCAT Slaves – status and address

      (e.g. Digital I/O modules, power modules, safety logic modules)

Hardware device Hierarchy in OPC-UA



Naming Convention

- Ideally each Data Items in CTA should have a unique name.
- We propose to follow a hierarchical approach to define the name of each data items.

Site.telescope.assembly.opc_ua_device.opc_ua_node_name.

Es.

CTAS.MST01.MOUNT.DVR.AZ_MOTOR01_VEL

CTAS.MST01.MOUNT.PWR.PhL_Current

For each DataItem: value, timestamp, quality (good, bad)

Next Steps

- Dedicated meeting with all the Telescope and Camera teams to enter in the details of what we discussed today.
- Definition of a abstract hardware device hierarchy for all telescope (and standard names and abbreviation for them) (slow do this proposal based on the information collected from the telescopes)
- Definition of an abstract model for sensors, digital I/O and actuators that are common to all hardware device
- Definition of the common Monitor, control, safety and configuration data items.
- Final definition of the naming convention

How to Organize the work

1. We suggest that after we agreed on the abstract hardware device hierarchy, the expert of each hardware device of each telescope team sit together (coordinated by ACTL-SLOW) to
    1. Define the abstract model for sensors, digital I/O and actuators that are common to all hardware device
    2. Define of the common Monitor, control, safety and configuration data items.
    3. Define an abstract model for UPC-UA information model for each Hardware device.

2. ACTL Slow and telescope teams will define the ICD tables (for each hardware device) that can then extended telescope by telescope to allocate specific Data Items. The ICD tables will take into account of both ACS and OPC-UA characteristics and will contain only definition of monitor, control, safety and configuration Data Items.

3. ACTL-SLOW will adapt the code generator to
    1. Develop a prototype application for each hardware device using the code generator including:
        1. OPC-UA server Information model. This will be the Hardware device simulator used to test the ICD
        2. ACS hardware device component and unit test.
    2. Telescope teams will test the prototypes
    3. Telescopes teams will extends the Hardware device to include the business logic related to each specific Hardware device.

These botton-up activities will be complemented by the top-down approach consisting in the final High level Interfaces at the Telescope and the development of the general level 0 ACTL-SLOW prototype.

This interface will be formalized in IDLs files in the ACS framework.

Schedule

- Next day we will circulate a possible schedule for the proposed activities in order to arrive to the Perugia F2F meeting, where:
  - we can summarize the status of this work and start Develop/testing the first hardware devices,
  - Work on and Test the first prototype of ACTL-SLOW software (limited to the implementation of the high level telescope interface and a basic Master Components).