The Control Software for the VIS instrument onboard of the ESA Euclid Mission

Emanuele Galli
emanuele.galli@iaps.inaf.it

On behalf of CDPU-SW Team
(A.M. Di Giorgio, M. Farina, G. Giusi, G. Li Causi, S. J. Liu, S. Pezzuto)
Euclid Mission

- Euclid mission has the aim to study **dark energy and matter** with unprecedented precision
- Wide field telescope to be place in L2 orbit
- Wide survey of the entire extra-galactic sky ($\geq 15,000 \text{ deg}^2$) with a deep survey ($\sim 40 \text{ deg}^2$)
- It is composed of two instruments:
  - *Visible Imaging instrument* (VIS)
    - High-precision galaxy shape measurements for the measurement of weak lensing shear
  - *Near Infrared Spectrometer and Photometer instrument* (NISP)
    - Necessary to derive the photometric redshifts (e.g. distance estimates to scale the absolute amplitude of the gravitational shear of each lensed galaxy)
- Operational lifetime 6.25 years
VIS instrument

- A **Focal Plane (FPA)** with 36 CCDs and 12 Readout (ROE) each one handling 3 CCDs. Each CCD is 4238*4132 px
- A **Calibration Unit (CU)** which provides uniform illumination for calibration purposes
- A **Shutter Unit (RSU)** for on demand occultation of telescope light
- A **Payload Mechanism Control Unit (PMCU)** which is in charge of distributing power to CU, driving RSU and monitoring temperature of FPA
- **A Control Data Processing Unit (CDPU)** which is in charge of monitoring and controls the instruments, performs data processing and transfer Science Data to the Space Craft
VIS Electrical Architecture

12 Spacewire links

MIL-BUS-1553B

2 Spacewire links (warm redundant)

1 Spacewire link (cold redundant)
• Implement the instrument data-handling functions
  - Receive Telecommands from the Control Data Management Unit (CDMU)
  - Send slow telemetry to the CDMU
  - Interpret and execute telecommands
  - Implement on-board time synchronization
  - Acquire and monitor housekeeping from all subsystems
    15 instruments during nominal operation
  - Manage and monitor VIS operating modes
  - Activate FDIR procedures
  - Control all subsystems
Functional Requirements of CDPU-OBSW (2)

- Science data-handling functions:
  - Command the synchronism to acquire science data from 12 ROEs
  - Use of a lossless compression algorithm
  - Compress the full FPA (~10 Gbit)
Non-functional Requirements

- Average compression ratio 2.8:1 (Maximum budget per day = 520Gbit)

- Compression and transfer of science data to MMU has to be performed in less than 278s

- Data loss due to 1-single bit error no more than 0.05%

- Internal VIS commanding accuracy better than $10^{-2}s$
VIS OBSW issues

• Necessity to start some tests campaign of compression algorithm without the real hardware
  - The final CPU architecture was decided only on April 2014 (PowerPC vs SPARC)
  - Maxwell board delivery takes almost 40 months
  - First delivery of Maxwell Board to INAF-IAPS laboratory at the end of 2015 with only the MIL-BUS-1553B interface

• Integration of driver started only at the end of June 2016

• Limited resources:
  - PowerPC at 400MHz (3 CPU redundant)
  - 256 MB of RAM
  - 8MB of EEPROM
• Wrapper to lower Real Time Operating Systems:
  - Tasks handling
  - Scheduling
  - Inter Process Calls (semaphores, mutual exclusion, message queues, signal, events, ...)

• Subsystem Interfaces strongly based on protocol/service layers

• Use of stubs for missing device drivers that emulates at least the load/delay
  - e.g. delay of transmission
Design software tool

- Use of a tool to model our application and to test the model itself → **IBM Rational Rhapsody**
  - Possible to generate code from UML state diagrams (event driven approach)
  - Allows to trace each easily user-requirements
Wrapper to Real Time OS

Core Services

Operating System Adaptor
Event processing

Event Allocated

Event in Memory Pool
Create

Event Allocated
Send

Event in Memory Pool

Event Sent
[Synch.] Execute

Event Sent
[Asynch.] Queue

Event Allocated

Event Handled
Destroy

Event Allocated

Event Handled
Handle

Event Allocated

Event Waits
Type of events and tasks interaction

Event

Pending Events

Destination

Current Event

EventQueue

Events Queue

Reactive

Dispatcher

TaskEM

<<singleton>>

TimerManager

Pending Timeouts

Timeout

Pending Events

<<singleton>>

Dispatcher

TimedAction

Timed Actions

Events Queue

Pending Events

Current Event

Dispatcher

Timed Actions
Example of use
Example of use (2)

New Event to process

Process of event depending on current state
What we were able to do...

End 2013

Windows Adapter on PC

Preliminary Compression Tests

2014

RTEMS Adapter on LEON board

Compression plus SpaceWire Tests

Mid 2014

VxWorks Adapter on GNU-simulator

OBSW running on VXWorks OS

2015/2016

VxWorks Adapter on Maxwell 750P

Tests on CPU board
... and what we can do now

- Already implemented most of expected functionality
- All interfaces tested adopting emulators of subsystems and spacecraft
Different types of testing:

- Static analysis: used to verify some metrics defined in the verification and validation plan
  - Coding standard
  - Percentage of comments
  - LOC per file
  - Cyclomatic Complexity
  - Level of nesting
  ...

- Unit Testing: executed to eliminate bugs at code or unit level
  - Each function is tested with a range of parameters
  - Each module is tested as an isolated item

- Model testing: verification of the behavior of each module as reaction to an external stimulus

- Requirement covering
Static Testing

- Static testing is performed using the **C++Test Tool**
  - Each rule can be configured (e.g. maximum cyclomatic complexity)
Unit Testing

- Unit testing is performed using the **C++Test Tool**
Model Testing

- Model testing can be performed directly inside the IBM Rational Rhapsody using the optional tool **IBM TestConductor**
  - Events can be configured and generated at a specific tick-time
  - Sequence diagrams can be generated to see if the behavior was as expected
• Use of IBM Ration Rhapsody with IBM Rational Rhapsody Gateway
• IBM Gateway is a gateway between the Software Requirement Document and the design as well as the code itself
  - Necessity to create a parser
  - IBM DOORS would have been the best solution

SMXF_LR.1.4.1.4:Handle Event

<table>
<thead>
<tr>
<th>Specification</th>
<th>Handling Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Path</td>
<td>smxf_Requirements::LowLevelRequirements::Generic Framework Services::Reactive Class::Handling Events::Handle Event</td>
</tr>
<tr>
<td>Covered by Test Case</td>
<td>Code_tc_takeEvent</td>
</tr>
<tr>
<td></td>
<td>Passed</td>
</tr>
<tr>
<td>Traced by</td>
<td>smxf::RiCReactive.takeEvent</td>
</tr>
</tbody>
</table>

SMXF_LR.1.4.1.5:Handle Triggered Operation

<table>
<thead>
<tr>
<th>Specification</th>
<th>Handling Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Path</td>
<td>smxf_Requirements::LowLevelRequirements::Generic Framework Services::Reactive Class::Handling Events::Handle Triggered Operation</td>
</tr>
<tr>
<td>Covered by Test Case</td>
<td>Code_tc_consumeEvent</td>
</tr>
<tr>
<td></td>
<td>Passed</td>
</tr>
<tr>
<td>Traced by</td>
<td>smxf::RiCReactive.consumeEvent</td>
</tr>
</tbody>
</table>
Thank You!