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 (>= 15,000 deg^2) with a deep survey (~40 deg^2)
- It is composed of two instruments:
  - *Visible Imaging instrument* (VIS)

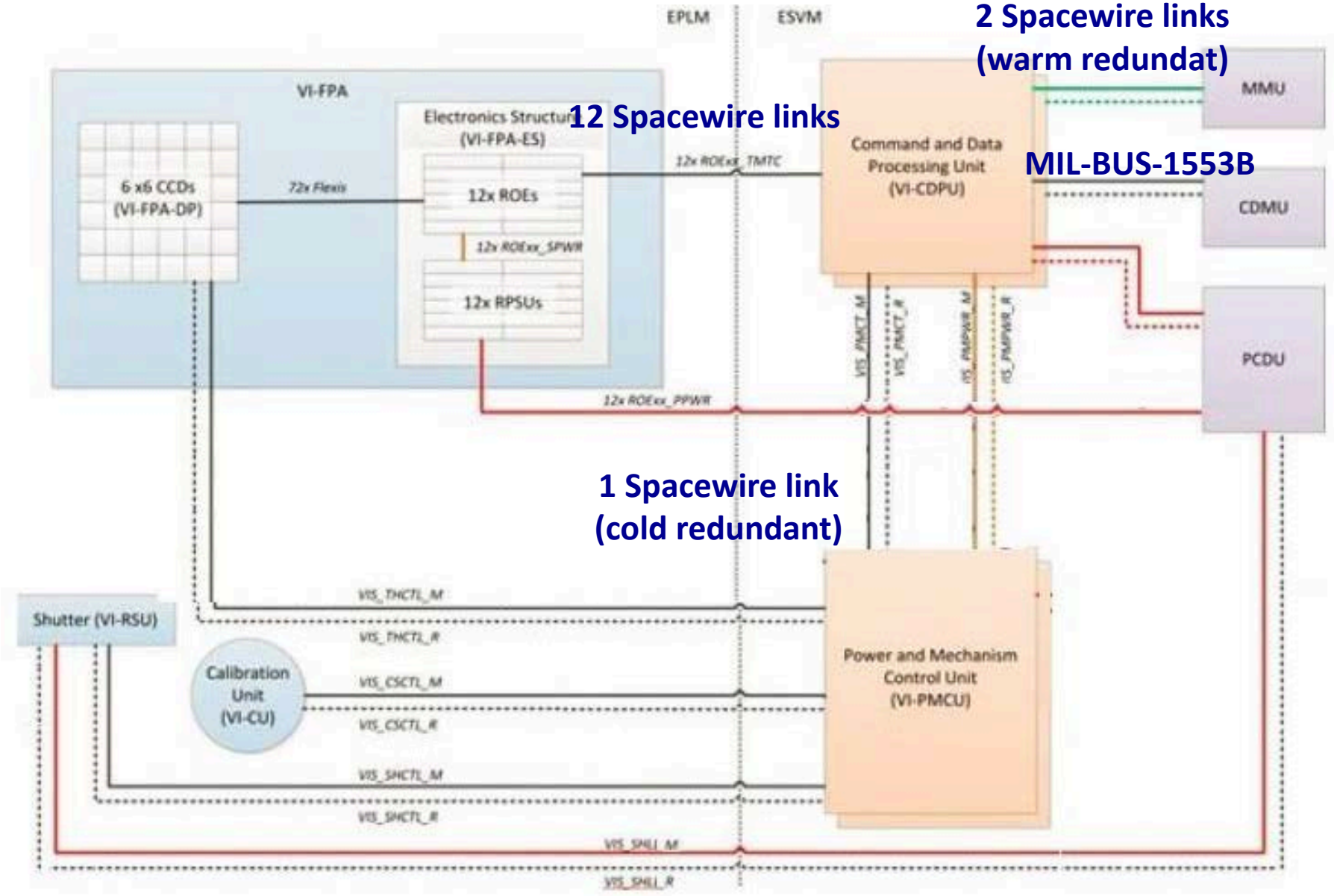
    High-precision galaxy shape measurements for the measurement of weak lensing shear
  - *Near Infrared Spectrometer and Photometer instrument* (NISP)

    Necessary to derive the photometric redshits (e.g. distance estimates to scale the absolute amplitude of the gravitational shear of each lensed galaxy)
- Operational lifetime 6.25 years

VIS instrument

- VIS instrument consists of:
  - 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



Functional Requirements of CDPU-OBSW (1)

- 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

VIS-OBSW solution: middleware multi-layer

- Wrapper to lower Real Time Operating Systems:
  - Tasks handling
  - Scheduling
  - Inter Process Calls (sempahores, mutual exclusion, message queus, signal, events, …)

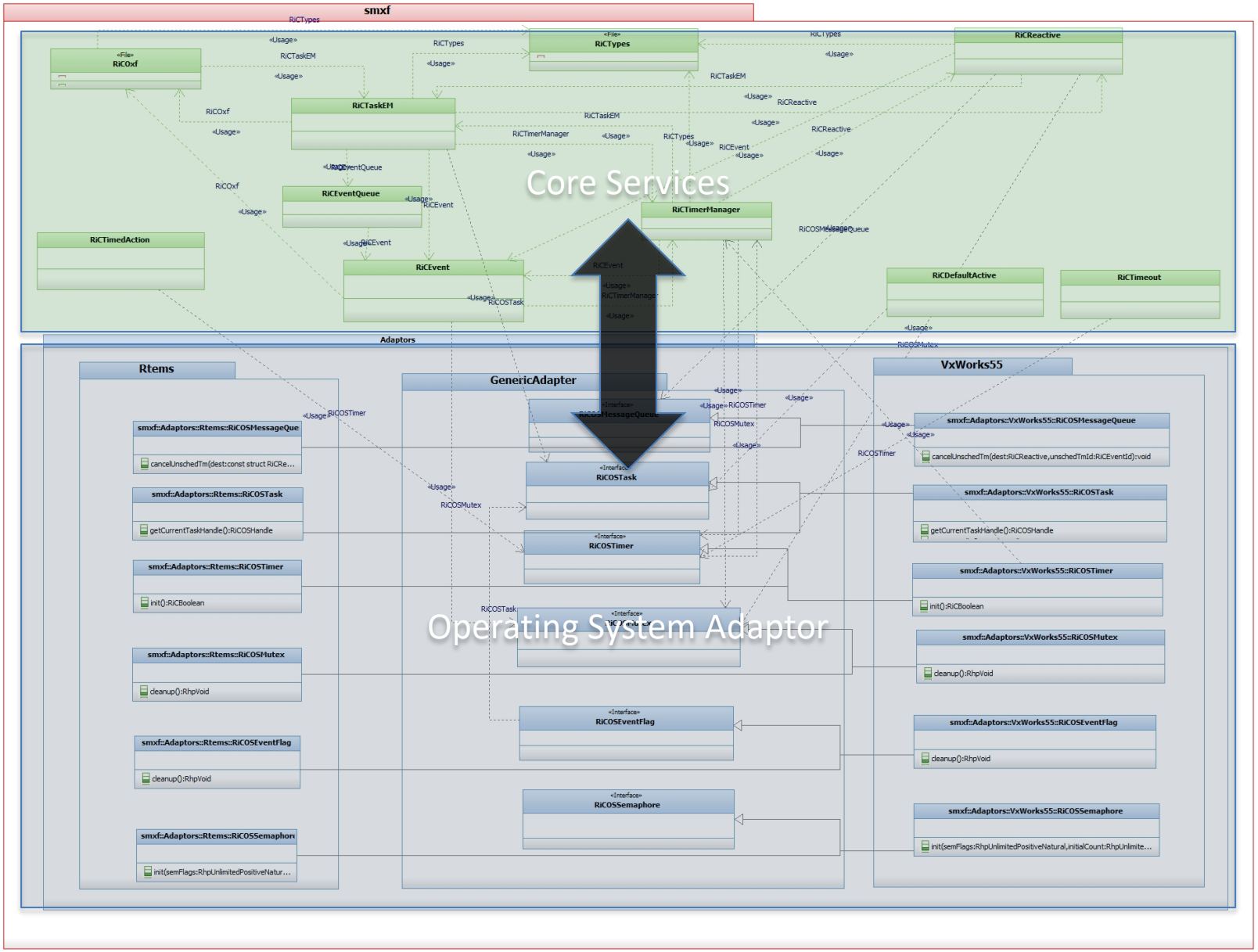
- Subsystem Interfaces strongly based on protocol/service layers

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

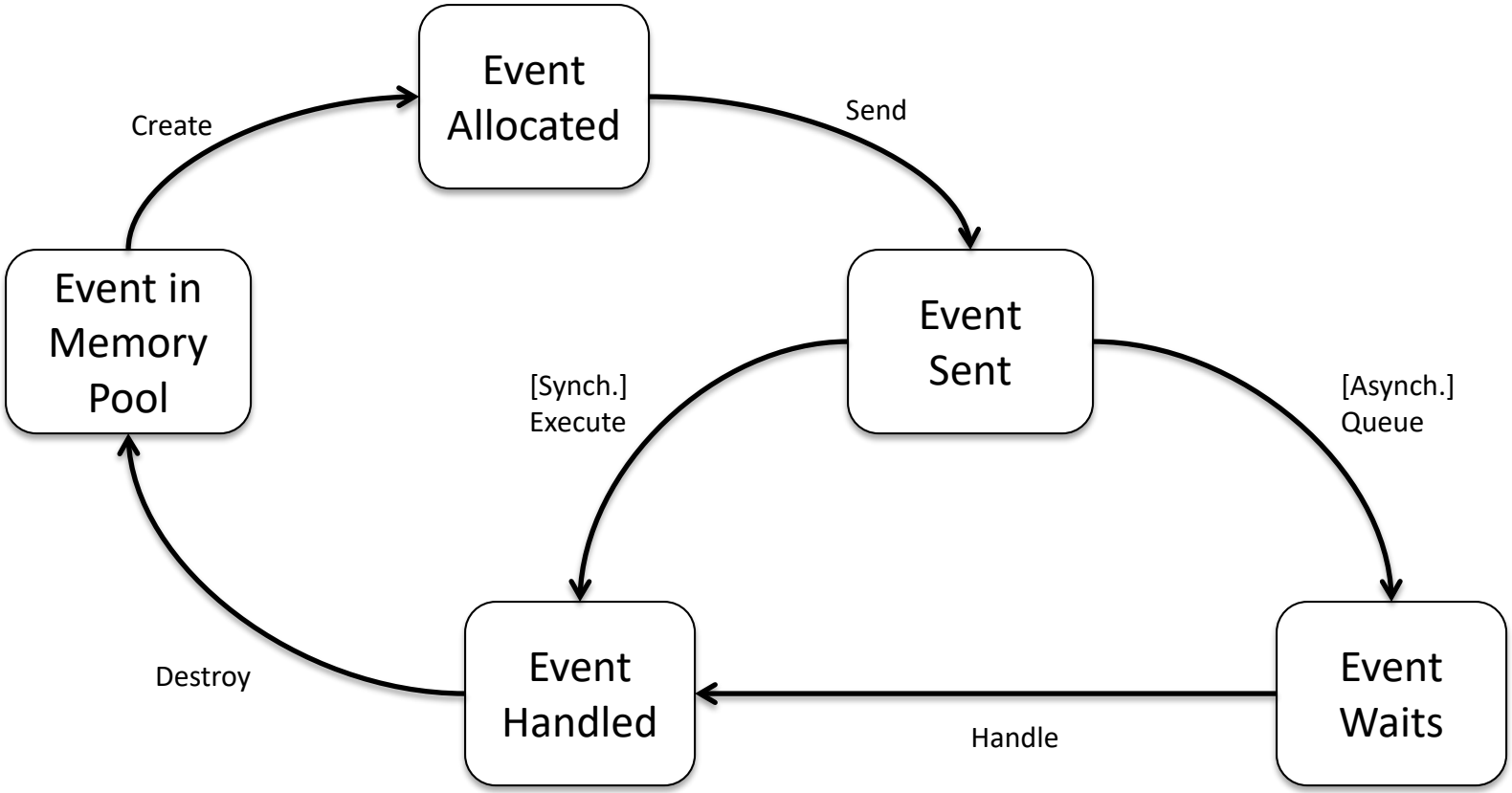
Design software tool

- 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

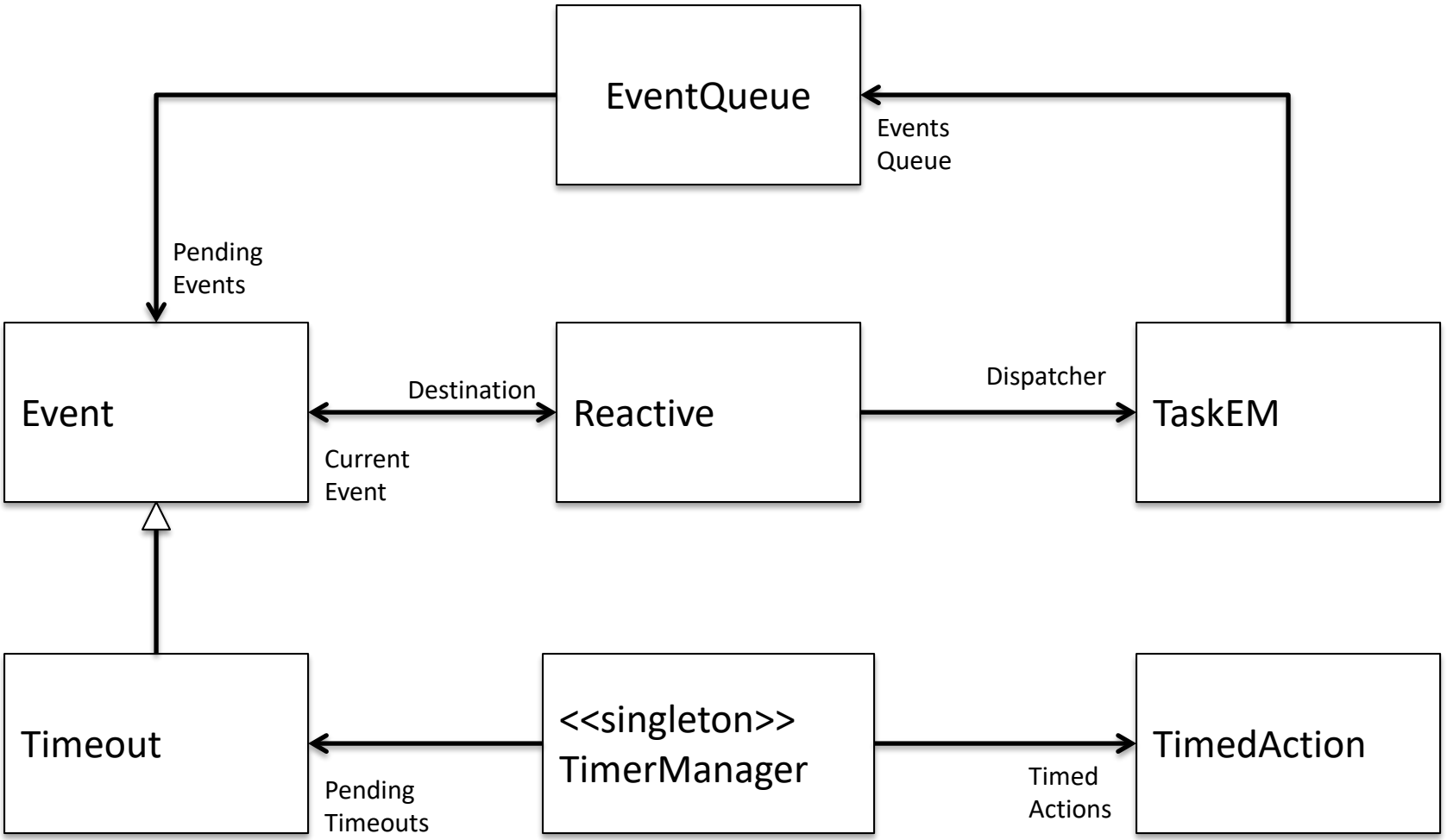
Wrapper to Real Time OS



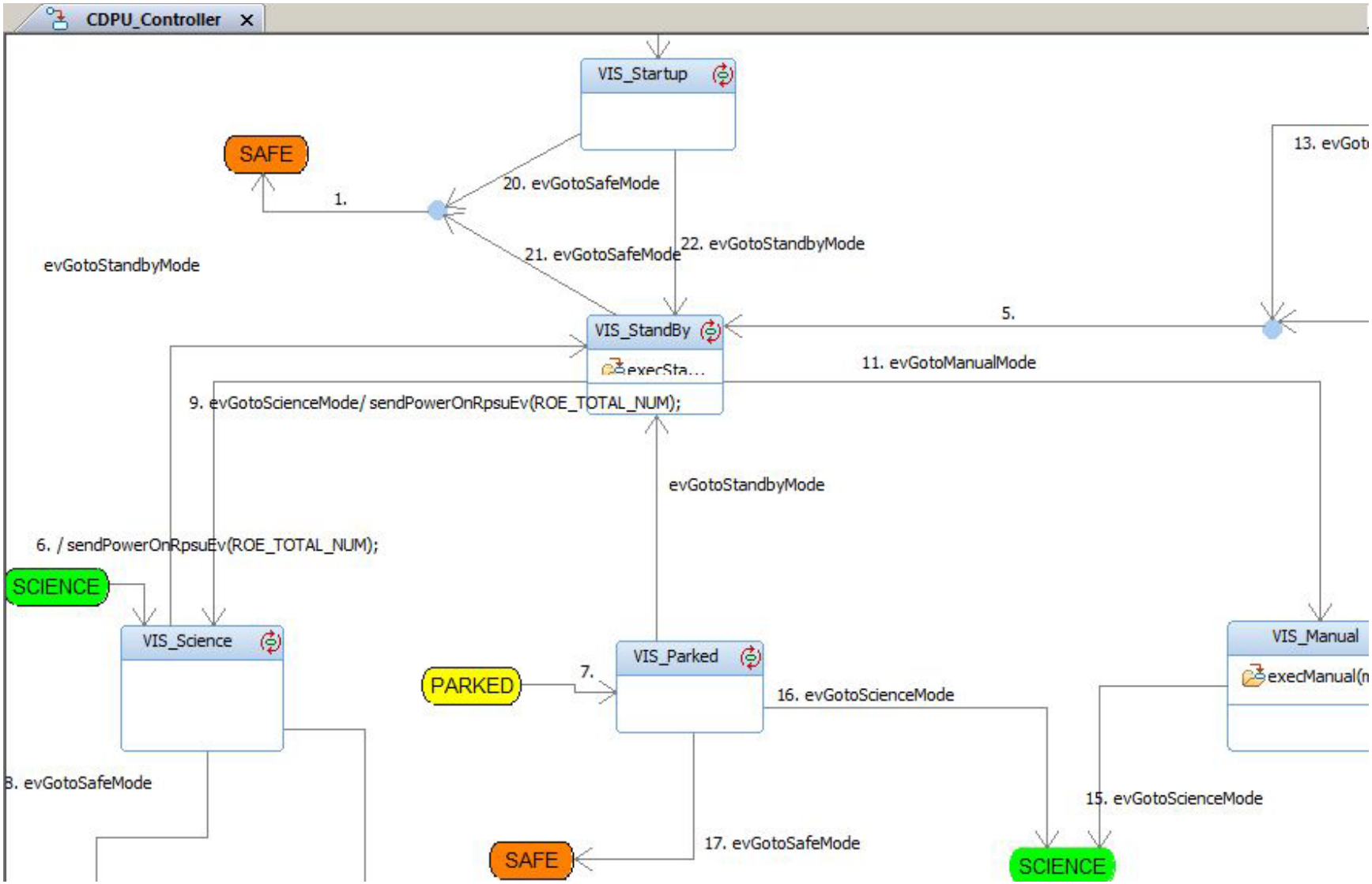
Event processing



Type of events and tasks interaction



Example of use



Example of use (2)

CDPU_Controller | CDPU_Controller.h | CDPU_Controller.c

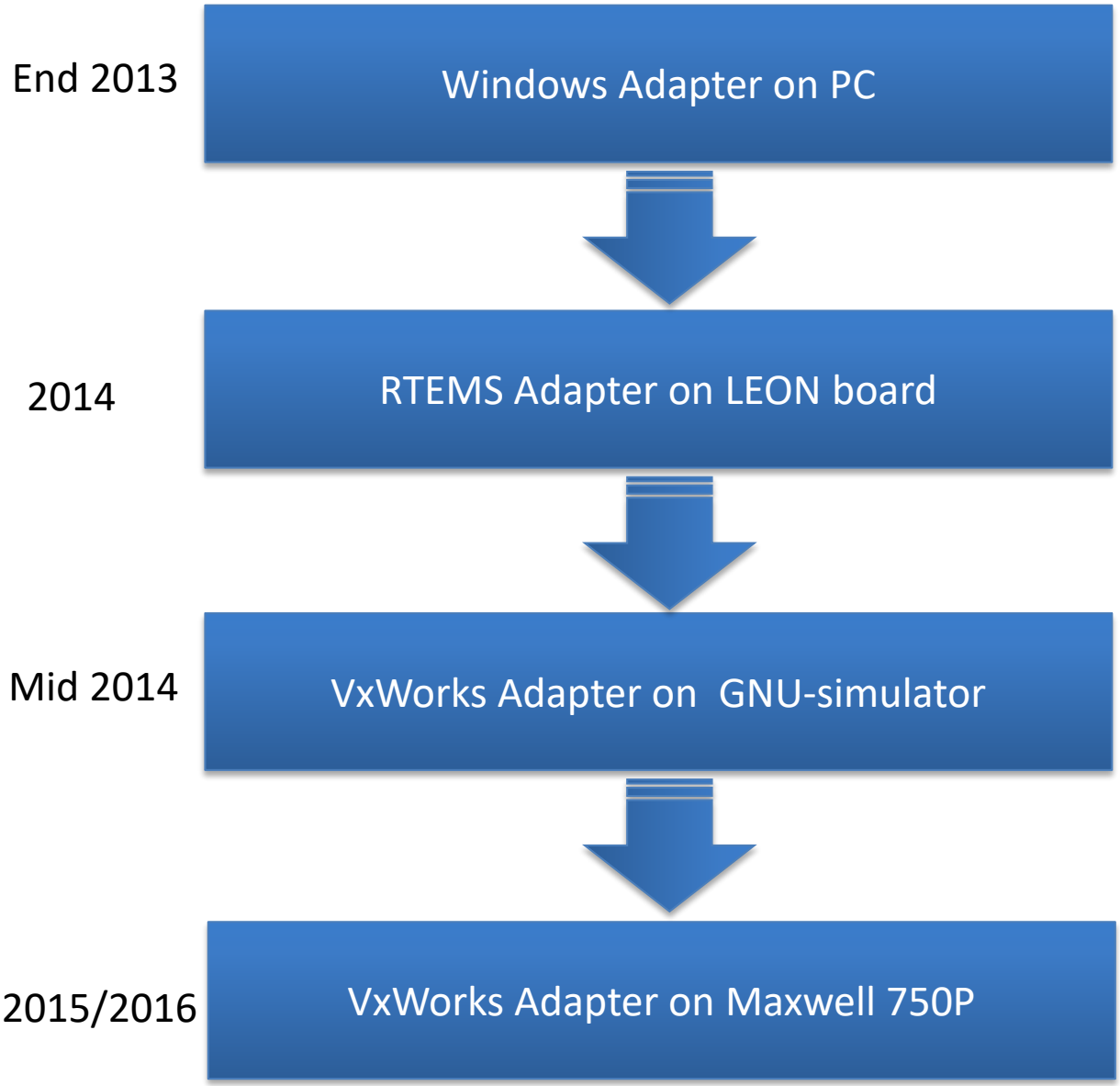
```
0193            RiCEvent * ev = NULL;
0194            do {
0195                /* Actually dispatch the event */
0196                { /* Access the event queue */
0197                    RiCTaskEM_lock(myTaskMember);
0198                    if (RiCOSMessageQueue_isEmpty(&(myTaskMember->eventQueue
0199                    {
0200                        /*Flag wait is blocking, then the mutex has to be fr
0201                        RiCTaskEM_free(myTaskMember);
0202                        RiCTaskEM_flagWait(myTaskMember);
0203                    }
0204                    ev = RiCOSMessageQueue_get(&(myTaskMember->eventQueue));
0205                    RiCTaskEM_free(myTaskMember);
0206                } /* mutex is freed */
```

New Event to process

```
1281                }
1282                break;
1283                /* State VIS_StandBy */
1284                case CDPU_Controller_VIS_StandBy:
1285                {
1286                    switch (id) {
1287                        /* Realizes requirement OBSRS-GEN-MT-0020 #OBSRS
1288                        /* The mode transition from VIS-Standby to VIS-S
1289                        case evGotoScienceMode_AppLayer_id:
1290                        {
1291                        {
1292                            /*#[ transition 9 */
1293                            sendPowerOnRpsuEv(ROE_TOTAL_NUM);
```

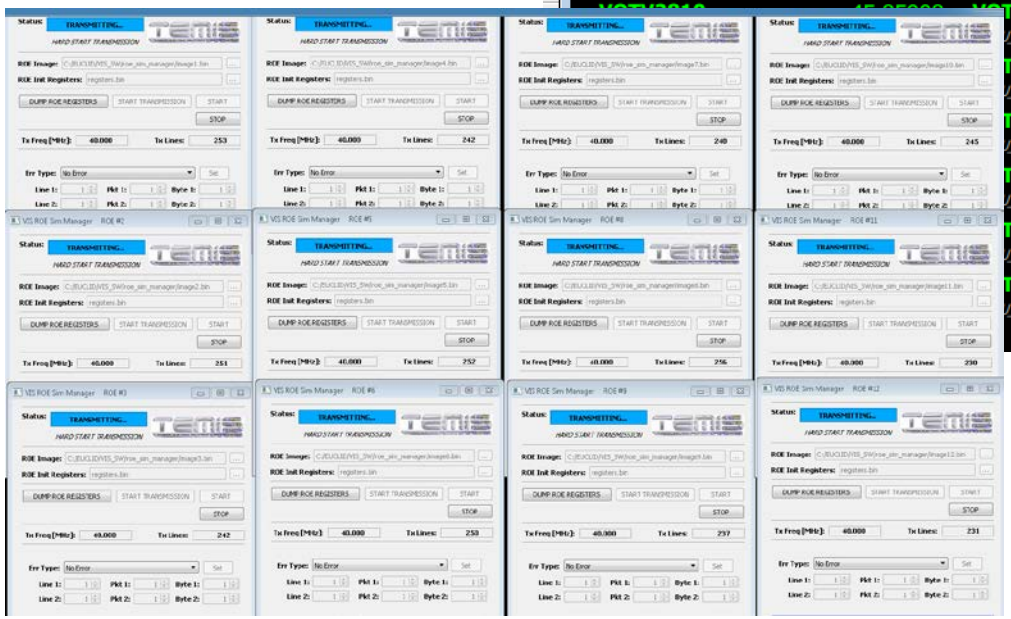
Process of event depending on current state

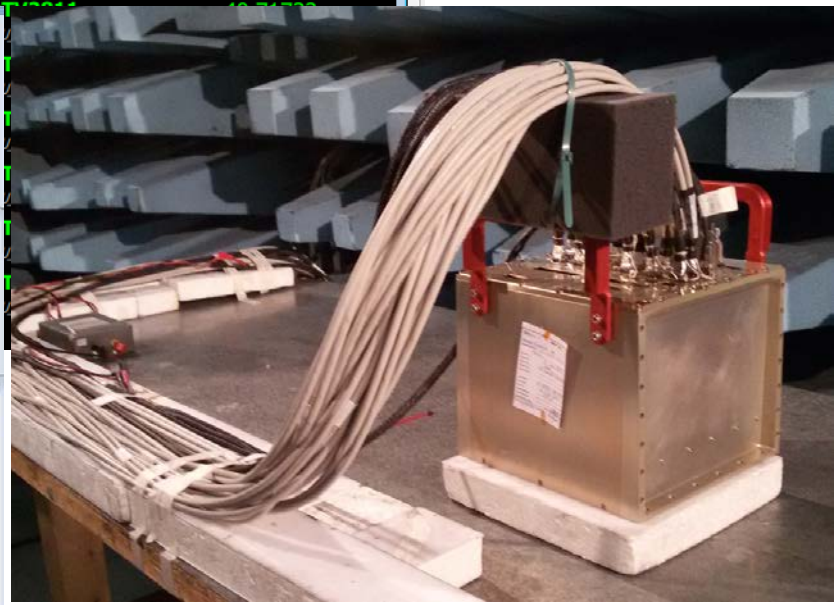
What we were able to do…



… and what we can do now

- Already implemented most of expected functionality
- All interfaces tested adopting emulators of subsystems and spacecraft





Testing of the OBSW

- Different types of testing:
  - Static analysis: used to verify some metrics defined int 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)

PARASOFT. C/C++test® Report

User configuration: **Metrics**
2016-11-17T11:00:58+01:00

STATIC ANALYSIS

Project Name	Tasks				Files		Lines	
	suppressed	qfix	total	per 10,000 lines	checked	total	checked	total
visdpu-giova	0	0	13	245	1	1	530	530
Total [0:00:13]	**0**	**0**	**13**	**245**	**1**	**1**	**530**	**530**

All Tasks by Category

by: Category Severity

[13] Metrics (METRICS)
[13] Report Cyclomatic Complexity (METRICS-29-5)

Tasks by Author

Back to Top

Author	Tasks		
	suppressed	Total	recommended
Euclid	0	13	13

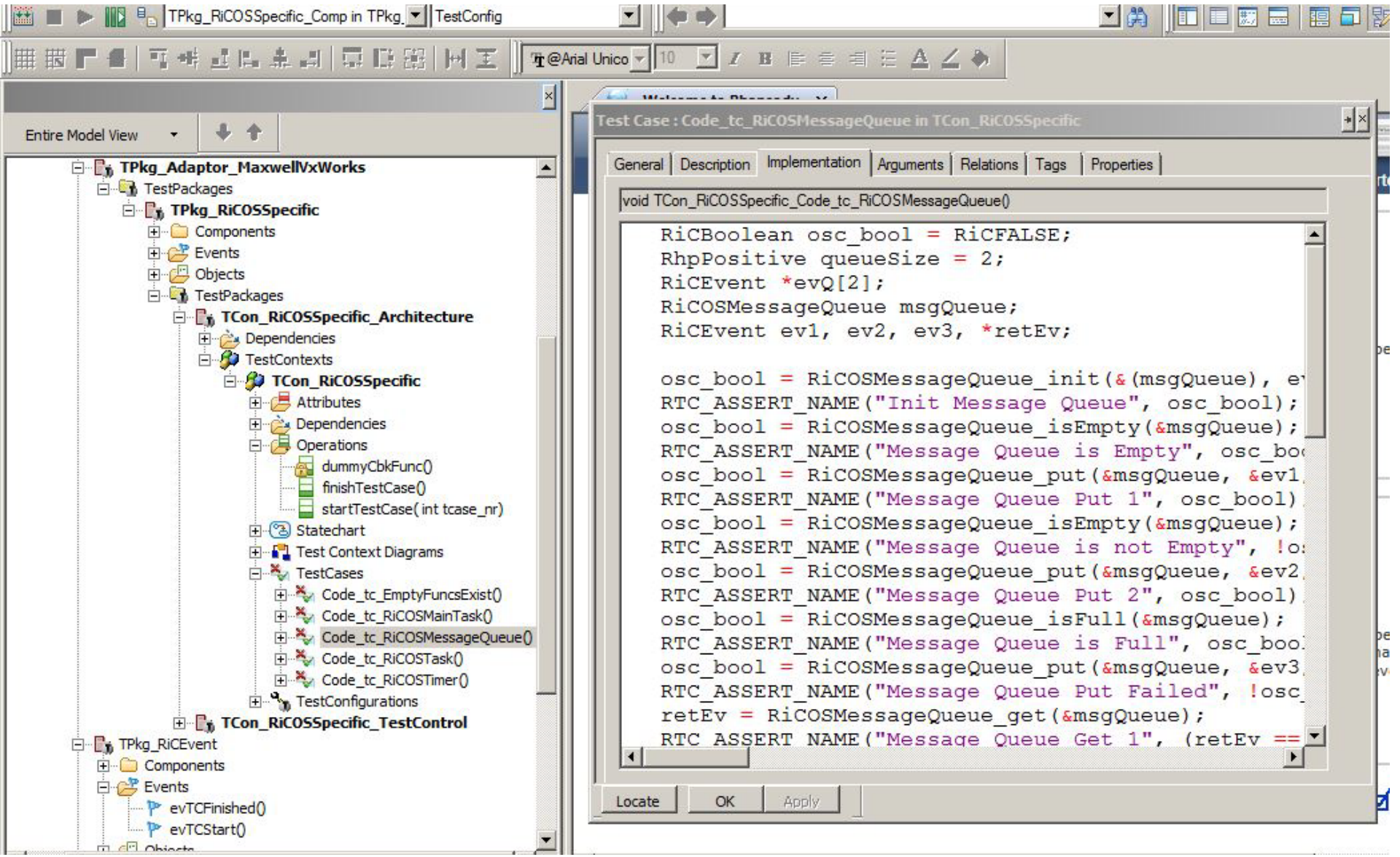
Active Rules

Back to Top

[0/26] BugDetective (License Required) (BD)
[0/1] Miscellaneous (BD-MISC)
[0/7] Security (BD-SECURITY)
[0/4] Resources (BD-RES)
[0/11] Possible Bugs (BD-PB)
[0/3] Threads & Synchronization (BD-TRS)
[0/4] Code Duplication Detection (CDD)
[0/252] Coding Conventions (CODSTA)
[0/91] Coding Conventions for C++ (CODSTA-CPP)
[0/13] Comments (COMMENT)
[0/18] Exceptions (EXCEPT)
[0/46] Formatting (FORMAT)
[0/18] Initialization (INIT)
[0/269] Joint Strike Fighter (JSF)
[3/60] Metrics (METRICS)
Follow the Cyclomatic Complexity limit of 20 (METRICS-28-3)
Report Cyclomatic Complexity (METRICS-29-5)
Follow the limit for Inheritance Depth (METRICS-ID-5)
[0/60] MISRA C (MISRA)
[0/214] MISRA C 2004 (MISRA2004)
[0/278] MISRA C++ 2008 (MISRA2008)

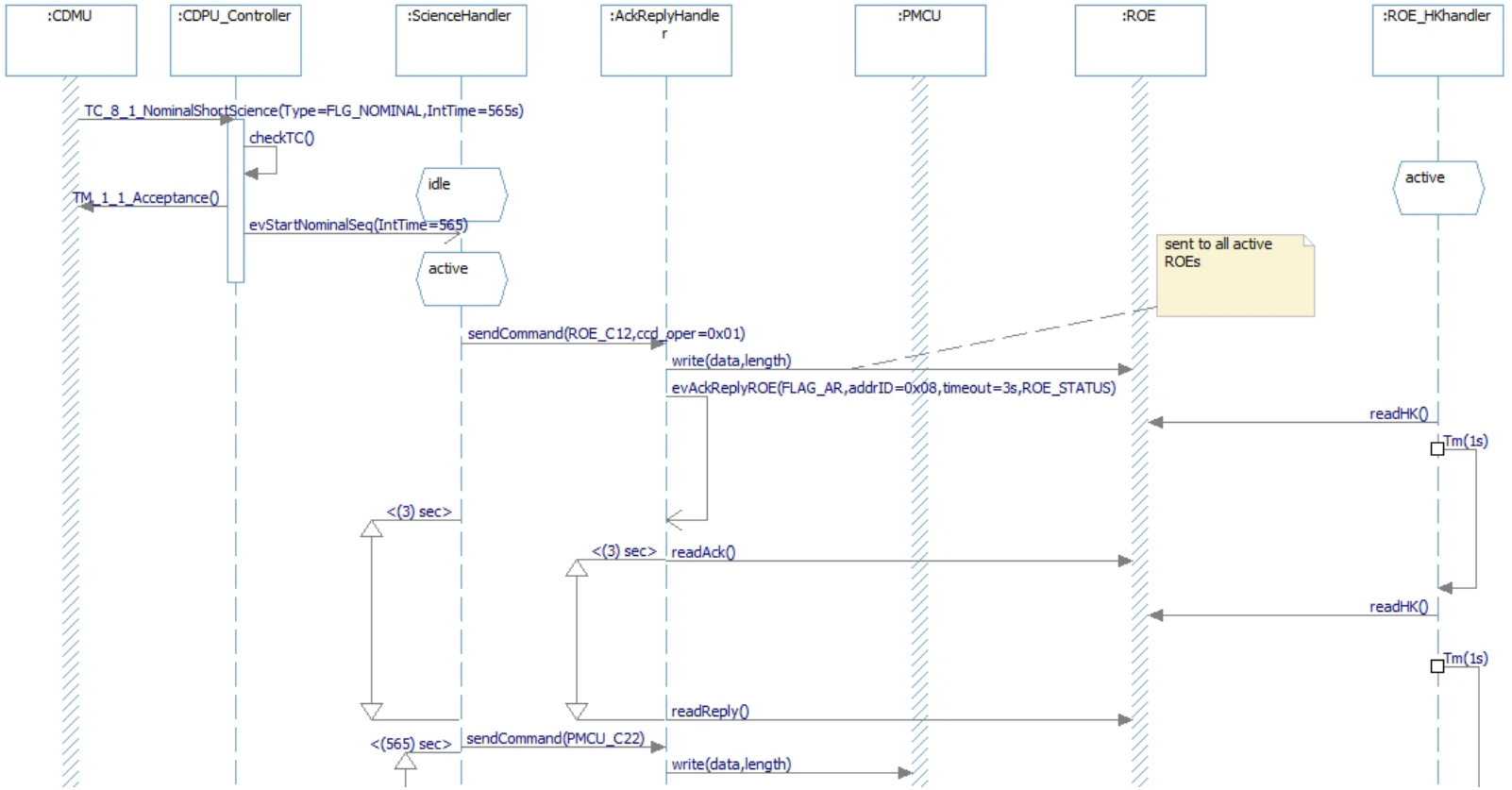
Unit Testing

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



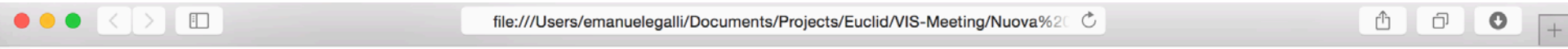
Model Testing

- 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



Requirements covering

- 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

Specification	This function shall process an event and return the status of operation. The function returns RiCTakeEventError if preconditions are not met. If this Reactive is in cleanup mode, RiCTakeEventInCleanup is returned and the event is discarded. If the Reactive was ordered to terminate, the event is ignored. The consumption of the Event is invoked and its result stored for return value. This invocation is guarded by eventGuard mutex, since the consumption of the events is a critical section, which can be accessed in parallel. If the Reactive should terminate following this consumption, the RiCTakeEventReachTerminate is returned.
Package	Handling Events
Full Path	smxf_Requirements::LowLevelRequirements::Generic Framework Services::Reactive Class::Handling Events::Handle Event
Covered by Test Case	Code_tc_takeEvent (Passed)
Traced by	smxf::RiCReactive.takeEvent

SMXF_LR.1.4.1.5:Handle Triggered Operation

Specification	This function shall handle a triggered operation event (synchronous event) and return the status of operation. The function returns RiCTakeEventError if preconditions are not met, otherwise, if this Reactive should terminate, the event is discarded. Otherwise, the event is processed immediately, by calling RiCReactive_consumeEvent. The result of this event consumption is returned.
Package	Handling Events

Thank You!