SPARTA Lessons Learnt, an Operational Perspective

M. Suárez Valles (ESO)

4th Adaptive Optics Real-Time Control Workshop

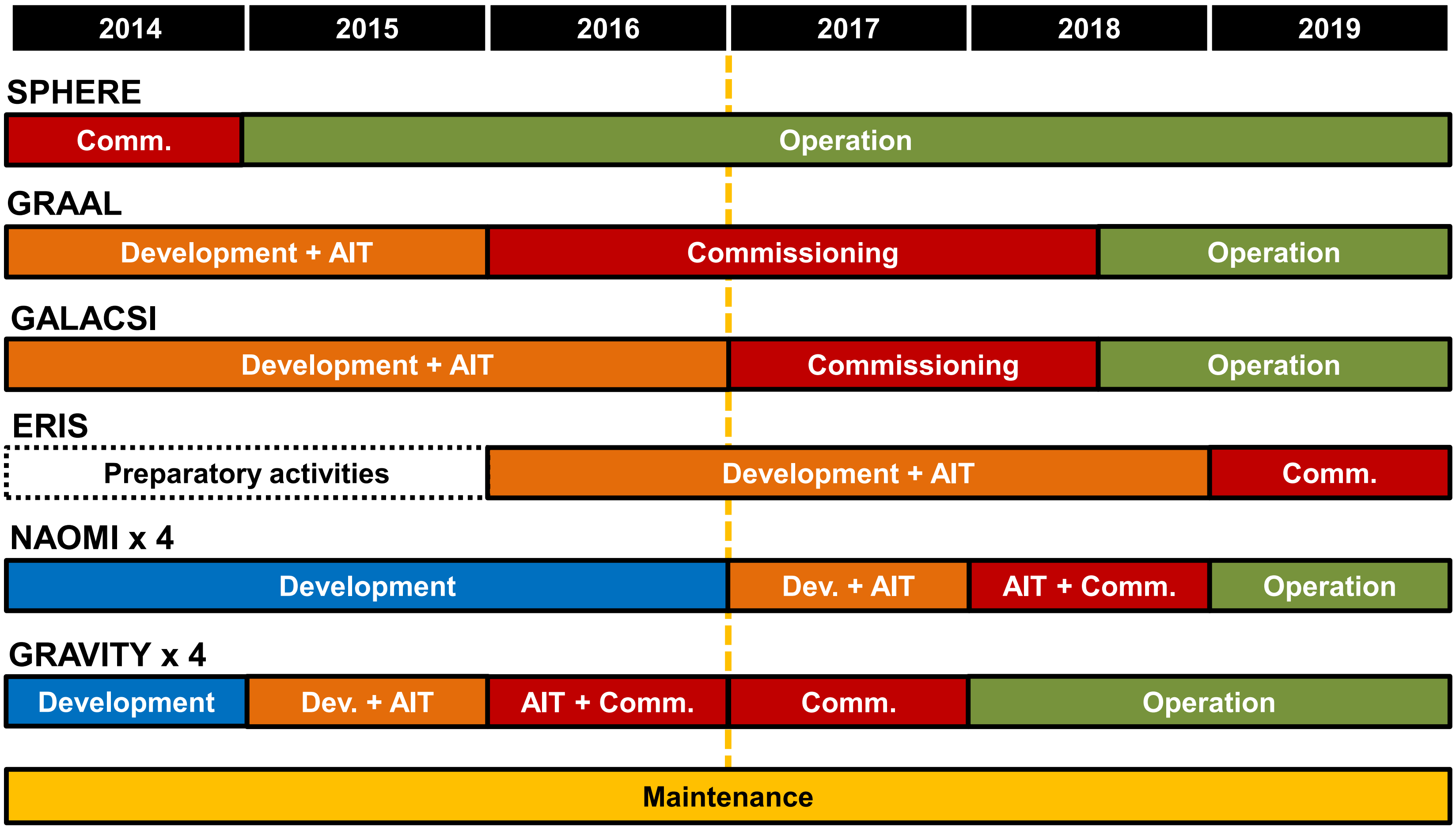
Observatoire de Paris (19/12/16 – 21/12/16)

Introduction to SPARTA

- ESO Standard Platform for Adaptive optics Real-Time Applications
- Conceived for supporting the 2nd generation VLT AO instruments:
  - SPHERE XAO module (SAXO)
  - AOF GLAO/LTAO modules (GRAAL, GALACSI)
- Later adopted for the implementation of:
  - ERIS SCAO module
- Extended through a lightweight version (SPARTA-Light) to serve:
  - New AO module for AT Interferometry (NAOMI) - 4 units
  - GRAVITY AO modules for the VLTI (CIAO) - 4 units
- Project lifetime already exceeds 10 yr.
  - Early activities in 2004
  - PDR in mid 2007; FDR end 2008
- RTC designed around a mixture of technologies:
  - Hard real-time using hybrid FPGA/DSP/CPU boards and VXS serial fabrics*
  - Soft real-time using mainstream Linux servers in a 1GbE cluster

(*) Regular VME CPU boards for SPARTA-Light

SPARTA Current Status



Some degree of maintenance is common to all RTC system at a given point in time

Maintenance - Intent

- Maximize SPARTA compatibility with evolving VLT SW and HW:
  - Allow systems under development to comply with standards upon delivery
  - Do not preclude regular instrument SW upgrade plan once in operation
  - Minimize the need for SPARTA-specific configurations in the IT spare HW pool
  - Prevent critical HW from using deprecated versions of OS and SW tools
- Maximize SPARTA operational life time:
  - Guarantee critical HW spare parts for the instrument's life time (~10-15 yr.)
- Simplify SPARTA-specific, on-site troubleshooting
- Follow up and solve operational issues

Bottom line: HW and SW obsolescence mitigation, to a great extent

Maintenance – Activities (1)

- Aligning SPARTA with new versions of OS and VLT SW:
  - Linux Kernel, device drivers, Linux toolchain
  - VxWorks kernel, cross-development toolchain
  - Common VLT libraries and tools

  e.g. Porting of sFPDP communication driver, overall porting to 64-bit architecture
  e.g. Porting of real-time control boards BSP to VxWorks 6.2 / 6.4 / 6.9

- Aligning SPARTA with new versions of SW products:
  - ACS, RTI DDS, Intel MKL, MATLAB

  e.g. Update to 64-bit ACS; regular DDS updates
  e.g. MATLAB upgrade for compatibility with Linux toolchain

- Adapting SPARTA to changes in licensing scheme of SW products

  e.g. Intel MKL no longer available as a standalone product

- SW bug-fixing and investigation of SPRs from systems in operation
- SW refactoring derived from systems still under development

Maintenance – Activities (2)

- Accommodating new HW standards into SPARTA:
  - IT server and network standards
  - VLT control standards

  e.g. Migration from rack to blade server format – 1 GbE vs. 10 GbE, RAID vs. network storage
  e.g. Potential VLT LCU obsolescence replacement (PowerPC → Intel)

- Shielding SPARTA from changes in evolving HW products

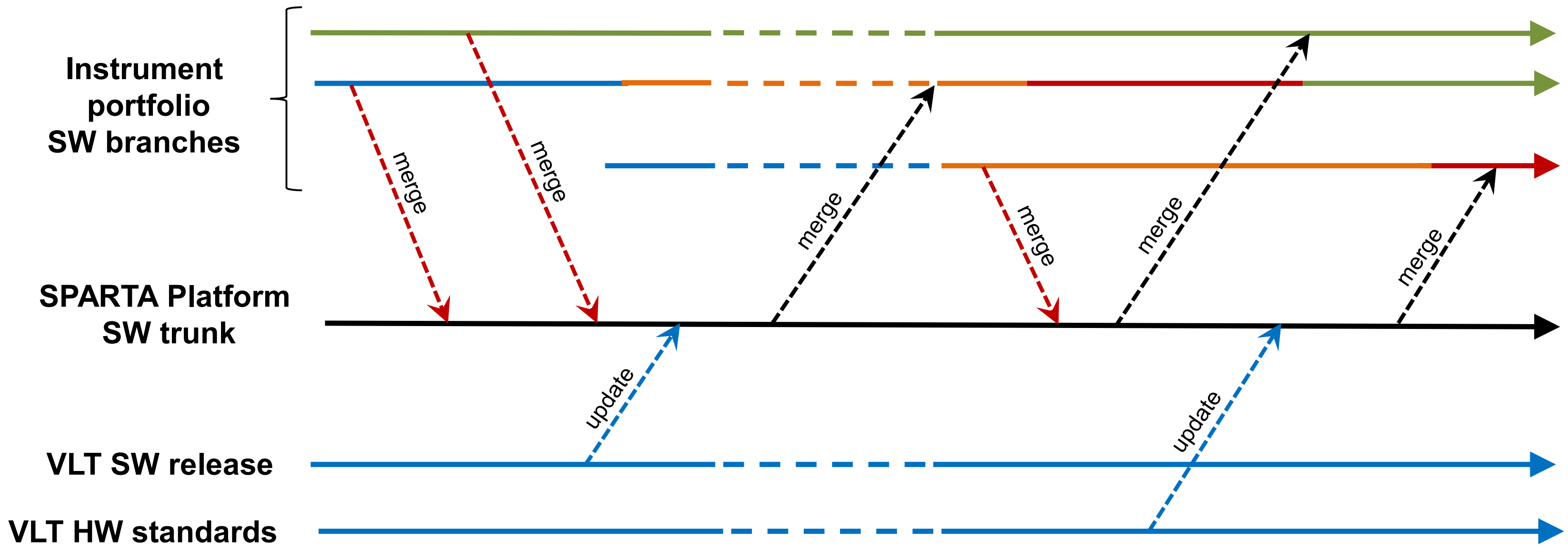
  e.g. Backwards-compatibility issues in newer revisions of the FPDP communication card

- Managing a spare parts pool for SPARTA critical HW:
  - Define pool size; monitor actual vs. predicted MTBF
  - Monitor product obsolescence and availability
  - Attempt repair of damaged units

  e.g. Spare parts pool refurbishment after Last Time Buy notice for hard real-time control boards

- Maintain SPARTA releases and installation procedure
  - Stable SPARTA for VLTSW2011 / VLTSW2014 / VLTSW2016
- Overall SPARTA documentation effort
- Training to new developers / Observatory staff

Maintenance – Process



- Maintenance requires comprehensive SW configuration control:
  - Diversity of instrument assemblies with frequent SW merging
- Maintenance is carried out in the absence of an actual controlled plant:
  - No dedicated AO bench; no sensors/actuators
  - No possibility for (synthetic) AO loop closure
- Instrument upgrades at any phase other than development require:
  - Difficult negotiation of the upgrade time slot
  - Re-commissioning or some form of performance re-assessment

Maintenance – Infrastructure

- <u>Testing</u> of the code base under maintenance is essential
- Extensive HW infrastructure is required for testing:
  - Full-scale, hard real-time control HW → may hinder spare parts pool
  - Full-scale, soft real-time cluster
- Some form of HW sharing becomes a must → virtualization
  - Multiplicity of instruments, each of them potentially under several OS versions
  - Hard real-time HW expensive (and obsolete)
- Continuous integration and testing SW infrastructure:
  - Automated, functional regression testing as part of periodic builds
  - On-demand, <u>numerical regression testing</u> based on MATLAB models
  - The test themselves require maintenance
- Efficient, numerical testing is a key factor:
  - To be performed with the hard real-time HW, without actual AO loop feedback
  - Requires input simulation and signal injection features from the RTC
- Maintenance to be carried out within limited FTE and budget

Lessons Learnt – Strategy & Scope (1)

- **An RTC platform pays off for maintenance** when targeted to a few number of instances, all defined within a limited time window:
  - Long-term maintenance FTE is drastically reduced:

2015	2016	2017	2018	2019
2.9	1.9	(2.05)	(1.5)	(1.10)

  - Difficulty in incorporating late-joiners -obsolescence, incompatible requirements
- **The RTC maintenance strategy must be consolidated early** during project setup
  - Otherwise difficult to secure commitments on FTE and budget
  - Observatory must be involved at all times and support the strategy
- **An RTC instance dedicated to maintenance must be costed early** during project setup
  - Otherwise maintenance strategy at risk and spare parts pool underestimated
  - A (close to) full-scale system proves necessary for meaningful testing
  - The test system itself requires HW maintenance, involves licensing costs, etc.

Lessons Learnt – Strategy & Scope (2)

- **Secure in-house resources for FPGA development**
  - Otherwise firmware maintenance is hindered after the contract is closed
  - The amount of firmware maintenance does not allow setting long-term contract
- **Consider carefully before basing a platform design on the single, most performant RTC instance**
  - Trade off with a (partially) dedicated design for the performance outlier
  - Challenge requirements… Constantly…
- **Avoid re-writing SW tools which are not RTC domain-specific**
  - Get Instrument/Observatory to extend/enhance and maintain existing tools
- **SW licensing schemes are to be closely followed** through the different project phases
  - Different licensing setups may apply to prototyping vs. development phases and maintenance/development vs. production systems
  - Geographical restrictions may apply: consider delivering to the Observatory in binary form for certain modules not requiring frequent rebuild
  - Yearly licensing is a significant fixed cost for maintenance: periodically evaluate open source alternatives –potential issues wrt. support, open project lifetime…

Lessons Learnt - Testing

- **Provisions for testability must be present in the RTC** already at early implementation phases
  - Signal injection/extraction at input and intermediate computing pipeline points... But also internal replay of simulated data at each stage (bypass physical I/Fs)
  - Only efficient way of developing on reduced systems / testing partial deliveries
- **Automate integration, functional and numerical testing**
  - Huge impact in the FTE required for testing during maintenance:

2015	2016	2017	2018	2019
0.45	0.50	(0.60)	(0.40)	(0.25)

- Achieving comprehensive, automated **RTC unit testing may not be a realistic expectation**
  - Distributed components require from common services and collaborations
  - Difficult to integrate real-time pipeline HW into unit (i.e. partial) test scenarios
  - Targeted system testing (incl. numerical) seems to guarantee correctness
- **A basic form of synthetic AO loop closure is desirable** but to be procured during the development phase
  - Unlikely to be approved during maintenance, once shown *"it can do without"*
  - Already a low frame-rate, non end-to-end facility would be extremely useful

Lessons Learnt – Design (1)

- **Allocate RTC hard and soft real-time functions to physically distinct subsystems** even at the expense of increased size
  - Virtually no soft real-time HW/SW obsolescence in ~10 yr.
  - Severe hard real-time HW obsolescence issues
- **Standardize all interfaces to the Instrument/Observatory SW** to be the same in all RTC instances
  - <u>Key feature</u> enabling future maintainability with limited FTE and HW systems
  - Standardize: command, configuration, data recording/injection, etc.
  - Do not expose hard real-time interfaces to the Instrument/Observatory SW
- **Do not over-simplify the RTC hard real-time interfaces**
  - It leads to duplicity of common services –e.g. command, configuration, logging
  - Evaluate a hard real-time implementation compliant with the soft real-time technologies –i.e. middleware
- **A hard real-time RTC pipeline as a *"flat"*, supervised pool of configurable DoF is probably not a realistic assumption**
  - The hard real-time will always need to encapsulate complex business logic
  - The soft real-time supervisory SW ends up being mostly a protocol adapter

Lessons Learnt – Design (2)

- **Minimize the number of hard real-time development environments and run-time target architectures**
  - Platform diversity adds maintenance and obsolescence risks
  - Select technologies compatible with foreseeable, long-term in-house expertise
  - Restrict FPGA usage to stable, performant functions requiring little maintenance
- **Single-source, niche real-time HW is prone to mid-term obsolescence**, even if commercialized in large yields
  - Lifetime and upgrade path controlled by very few, large customers
  - Ability to repair past the end of product lifetime is poor
  - Longevity of Supply/Repair plans are available but expensive
- **Aerospace/military/ruggedized HW is reliable**
  - Observed MTBF lower than predicted under almost 100% duty cycle
- **High-speed, backplane electrical serial interfaces are not necessarily *“plug’n play”* technology**
  - Signal integrity is to be tuned and DoF not always available/accessible all the way down from sensor, through RTC, to actuator
  - Requires domain-specific knowledge –some issues not yet fully understood
  - Consider alternatives and trade them for compactness

Thanks!

Questions?