Reaching into space
TOGETHER

Spacecraft On Board Software

November 2020

Copyright © 2017 by SPACEBEL – All rights reserved

- On Board Software
  - Overview
  - Characteristics
  - Processes
  - Architectures
  - Environments
  - Trends

Copyright © 2017 by SPACEBEL – All rights reserved

- On Board Software
  - Overview
    - Missions
    - Functions
    - Interfaces
    - Overall Architecture
  - Characteristics
  - Processes
  - Architectures
  - Environments
  - Trends

Copyright © 2017 by SPACEBEL – All rights reserved

- On Board Software
  - Characteristics
    - Constraints
    - Criticality
    - Dependability
    - Safeguarding
    - Properties
  - Processes
  - Architectures
  - Environments
  - Trends

Copyright © 2017 by SPACEBEL – All rights reserved

- On Board Software
  - Characteristics
  - Process
    - Phases
    - Lifecycle
    - Verification
    - Standards
    - Documentation
  - Architectures
  - Environments
  - Trends

Copyright © 2017 by SPACEBEL – All rights reserved

- On Board Software
  - Characteristics
  - Process
  - Architectures
    - Functional Architecture
    - Static Architecture
    - Interfaces & Data Flows
    - Dynamic Architecture
    - Deployement Architecture
  - Environments
  - Trends

Copyright © 2017 by SPACEBEL – All rights reserved

- On Board Software
  - Purposes
  - Characteristics
  - Processes
  - Architectures
  - Environments
    - Development Environment
    - Validation Environment
    - Execution Environment
  - Trends

Copyright © 2017 by SPACEBEL – All rights reserved

An On Board Software to Rule Them all

Copyright © 2017 by SPACEBEL – All rights reserved

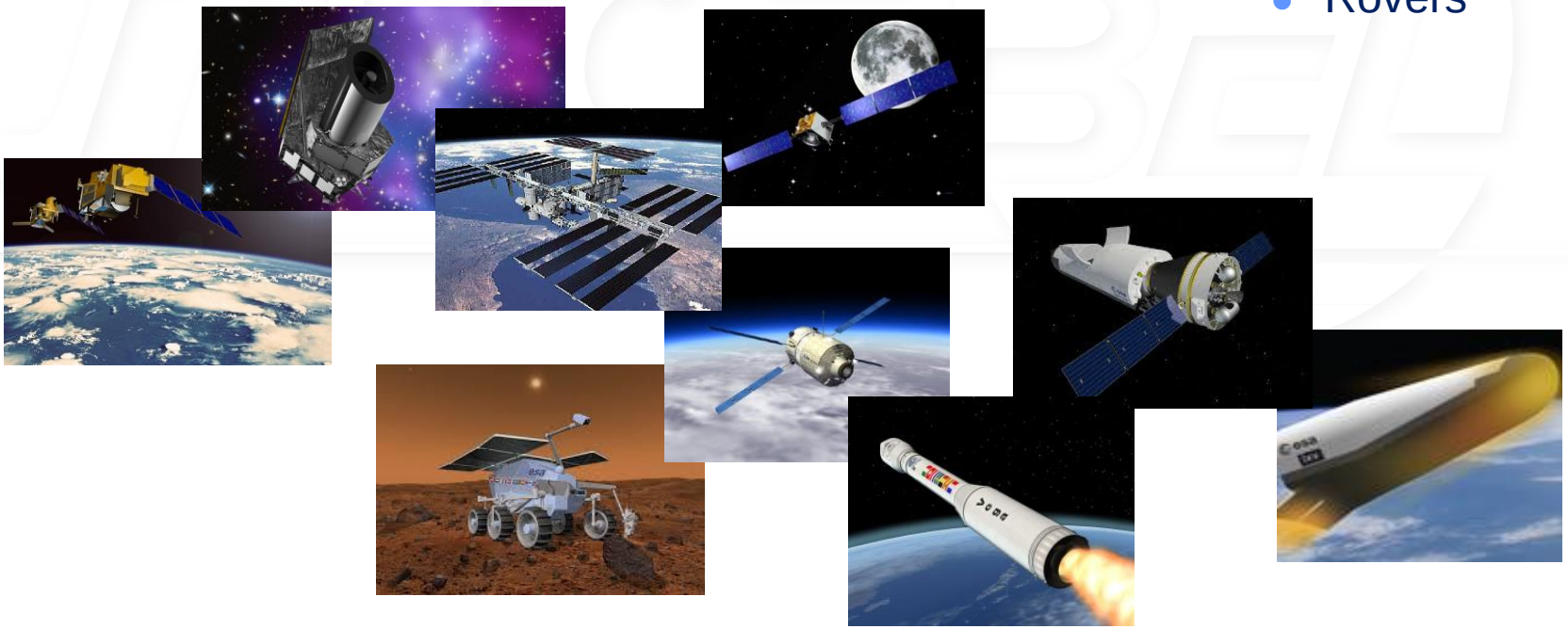
Overview

- Purposes
  - Missions
  - Functions
  - Overview

Copyright © 2017 by SPACEBEL – All rights reserved

Missions

- Missions
  - Earth Observation
  - Exploration
  - Science
  - Meteo
  - Communication
  - Radionavigation
- Platform
- Payload
- Spacecrafts
  - Satellites
  - Space Station
  - Space Probes
  - Launchers
  - Vehicules
  - Rovers



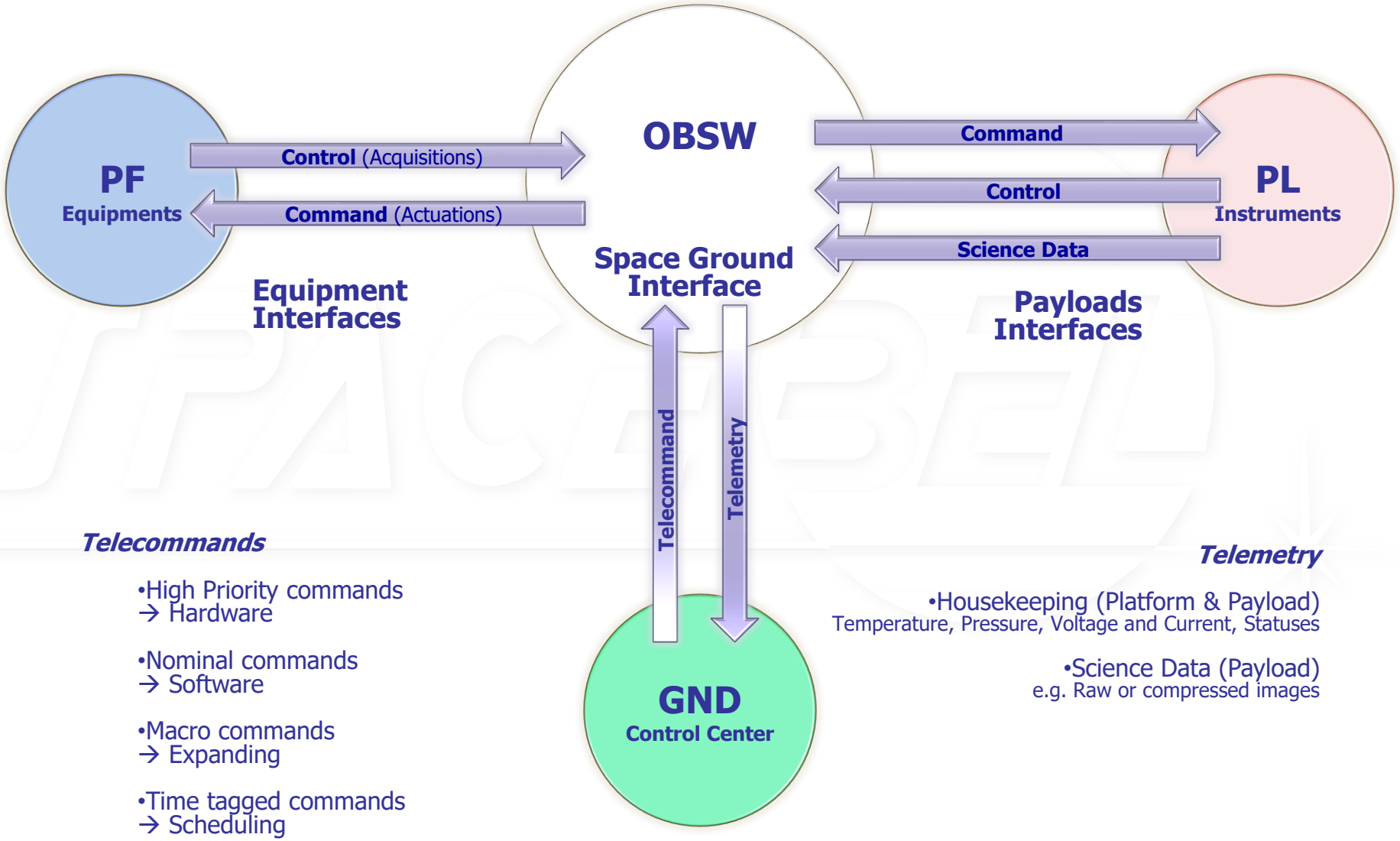
Copyright © 2017 by SPACEBEL – All rights reserved

Functions

- **Command, Monitoring and Control**
  - Observability, Commandability, Autonomy
  - **Operations**
    - Ground command and Control through Telecommands and Telemetries
  - **Platform Management**
    - Thermal, Power, Communication, Antenna, Attitude and Orbit Control subsystems
  - **Equipment Management**
    - Command and Control, Configuration, Fault Detection, Redundancy
  - **Payload Management**
    - Command and Control, Configuration
  - **Data Management**
    - Acquisition, Storage, Download
  - **Mode Management**
    - Mission Phases, Spacecraft Operational Modes, Equipment States
  - **Mission Timeline**
    - Time Triggered or Position Based Actions

Copyright © 2017 by SPACEBEL – All rights reserved

Interfaces & Data Flows



Telecommands

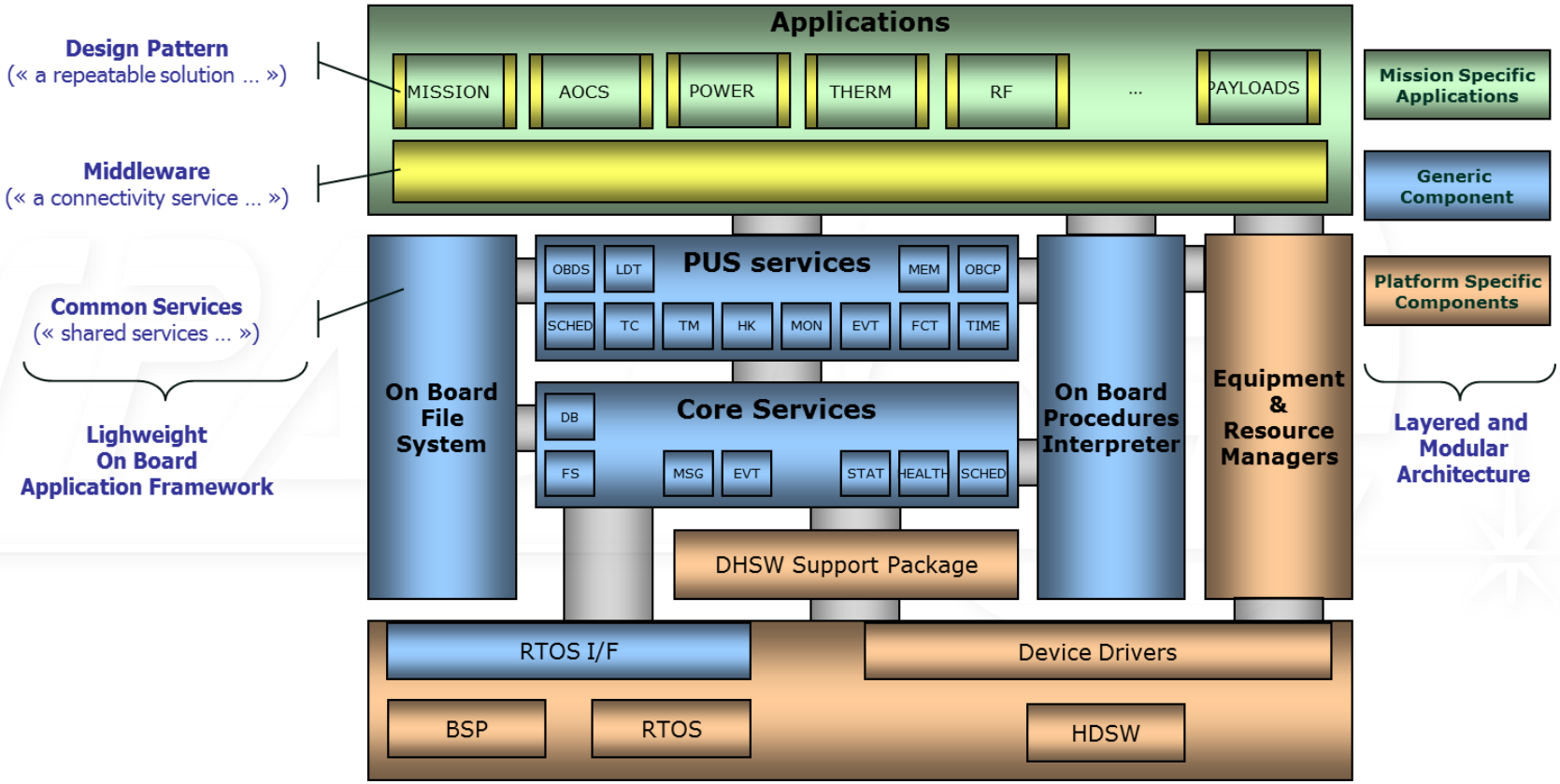
- High Priority commands → Hardware
- Nominal commands → Software
- Macro commands → Expanding
- Time tagged commands → Scheduling

Telemetry

- Housekeeping (Platform & Payload)
  Temperature, Pressure, Voltage and Current, Statuses
- Science Data (Payload)
  e.g. Raw or compressed images

Copyright © 2017 by SPACEBEL – All rights reserved

Overall Architecture



Copyright © 2017 by SPACEBEL – All rights reserved

The Characteristics

- Constraints
- Criticality
- Dependability
- Safeguarding
- Properties

Copyright © 2017 by SPACEBEL – All rights reserved

Constraints

On Board Software has to cope with constraints stemming from its execution environment in space:

- **Embedded** Software
  → Cross Development Environment, Constrained Execution Environment (limited Memory and Processing)
- **On Board** Software
  → Single Event Effects (Upset, Latch Up), Memory Scrubbing
- **Real Time** Software
  → Processor load, tasking issues, deadlines
- **Deterministic** Software
  → No dynamic thread creation or memory allocation, budget and schedulability analysis
- **Remote** Software
  → Need for autonomy (all the more for deep space)
- **Critical** Software (see further down)
  → Possible catastrophic consequences of failure
- **Dependable** Software (see further down)
  → Need for high Reliability, Availability, Maintainability and Safety

Copyright © 2017 by SPACEBEL – All rights reserved

Properties

Software is part of the System

Software requirements are derived from System requirements

Software properties must be derived from System properties.

e.g.

- End-to-end system response time will result into a
  → software schedulability property.
- System availability property will result into
  → Software safeguarding mechanisms.
- System performance property may result in a
  → Software numerical accuracy property.

Copyright © 2017 by SPACEBEL – All rights reserved

Criticality

Critical Software
is a Software that if not executed or if not correctly executed or whose anomalous behaviour could cause or contribute to a System Failure resulting in :

A:	**Catastrophic Consequences**	Loss of Life, life threatening, personnel injuries, permanently disabling injury or occupational illness, Loss of an element of an interfacing manned flight system. Damage to other equipment. Loss of launch site facility facilities or loss of system. Severe detrimental environmental effects.
B:	**Critical Consequences**	Permanent or non-recoverable loss of the satellite's capability to perform its planned mission Temporarily disabling but not life-threatening injury or occupational illness. Major damage to flight system or loss or major damage to ground facilities. Major damage to public or private property or major detrimental environmental effects.
C:	**Major Consequences**	Negligible or minor effect on the satellite's mission and operability A detailed definition is left on a project by project basis and reported in its risk policy. Example is Mission Simulation Software
D:	**Minor Consequences**	A detailed definition is left on a project by project basis and reported in its risk policy. *Example is Test Software*

*In the **DO-178-C** (Software Considerations in Airborne Systems),*
*the Design Assurance Level (**DAL**) is determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system.*
The failure conditions are categorized by their effects on the aircraft, crew, and passengers (catastrophic, severe/hazardous, major, minor, no effect).

Copyright © 2017 by SPACEBEL – All rights reserved

Dependability

Troubles

•**Error:**
A wrong or missing action
•**Fault:**
An incorrect step, process or data definition in a program
•**Failure:**
The inability of the software to perform its required functions

Strategies

•**Fault Prevention**
avoidance and reduction of fault causes
•**Fault Tolerance**
avoidance and reduction of fault consequences
•**Fault Removal**
removal of fault occurrences
•**Fault Forecasting**
prediction of behaviour in presence of faults

Analysis

•**SCA**:
Software Criticality Analysis (see next slide)
•**HSIA**:
Hardware Software Interaction Analysis
•**FMECA**:
Failure Mode Effects and Criticality Analysis
•…

Achieving mission objectives and ultimate mission success relies on dependability of the space systems and of the software.

As software plays more and more a prominent role in space systems, its contribution to the overall system dependability becomes a vital aspect of system development

RAMS

•**Reliability**:
continuity of correct service.
•**Availability**:
readiness for usage.
•**Maintainability**:
easiness of repair/upgrade.
•**Safety**:
non-occurrence of catastrophic failure

FDIR

•**Fault**
hardware or software
•**Detection**
e.g. through monitoring
•**Isolation**
determination of the cause
•**Recovery**
e.g through redundancy

•**Safety**
Protection against Hazards
•**Security**
Protection against Threats
•**Integrity**
Maintenance of consistency

•**Certifiability**
Ability to get stamp from certification body

FDIR is imperative to guarantee a dependable and autonomous systemwith a minimal risk of ruinous failure

Copyright © 2017 by SPACEBEL – All rights reserved

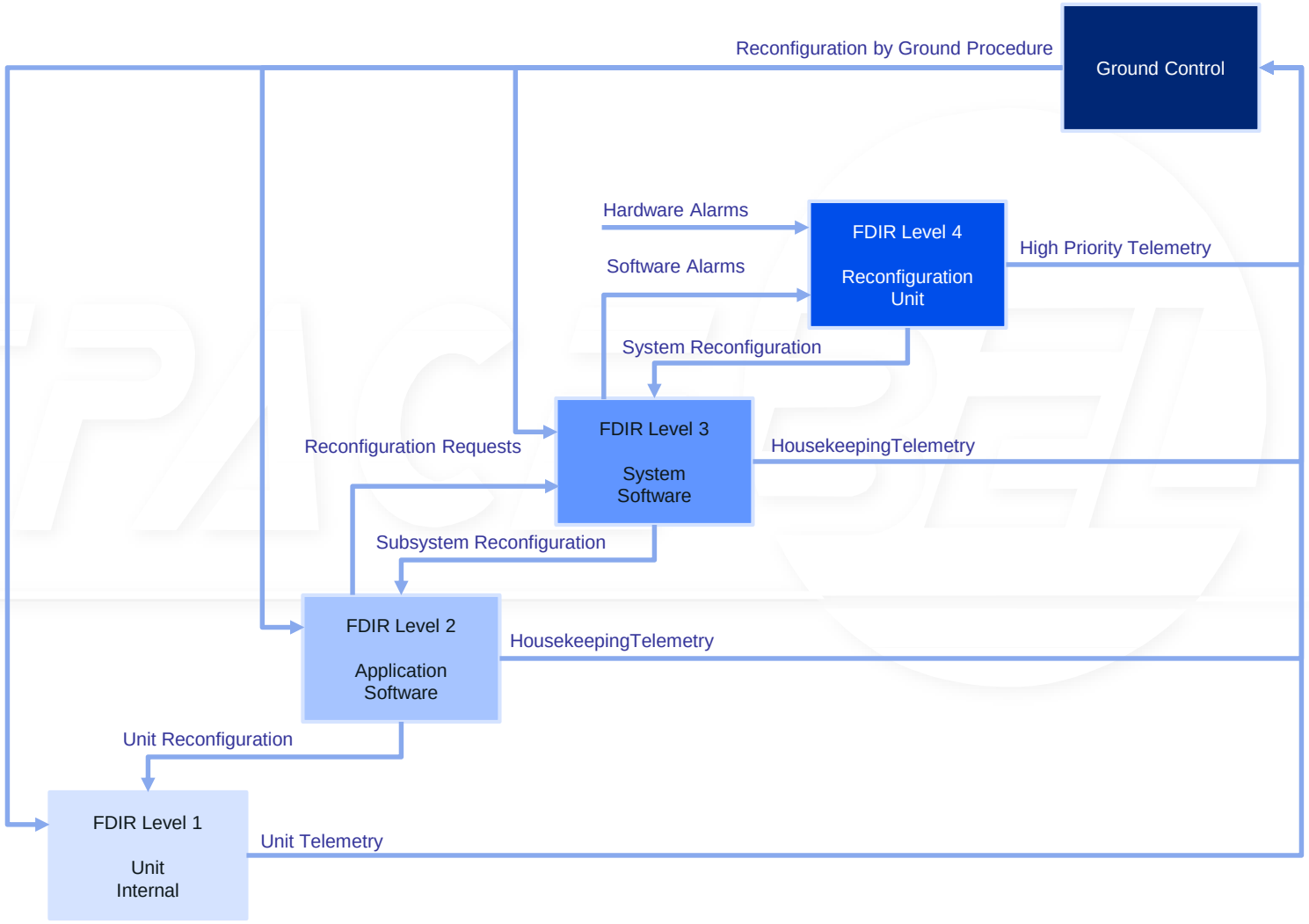
Safeguarding

Level 4	Handled by Operators on **GND**	Major Overall System Failure Communication Failure, Deployment Failure …
Level 3	Handled by Reconfiguration Unit in On Board Computer **HW**	Hardware Induced Alarms Multiple EDAC alarms, S/C Power Failure …
Level 2	Handled by Spacecraft **System SW**	System Malfunction Attitude Computation Inconsistencies, …
Level 1	Handled by Unit Manager **Subsystem SW**	Subsystem Malfunction Subsystem Equipment or Intercommunication Failure …
Level 0	Handled Internally by Unit (transparent to SW)	Unit Internal Malfunction internally recoverable, requiring instant reaction shiort current protection, bus retries, single EDAC …

FDIR design relies on a clear **hierachy** defining which **type of failure** is to be identified and managed on **which level**

Copyright © 2017 by SPACEBEL – All rights reserved

Safegarding (cont.)



Copyright © 2017 by SPACEBEL – All rights reserved

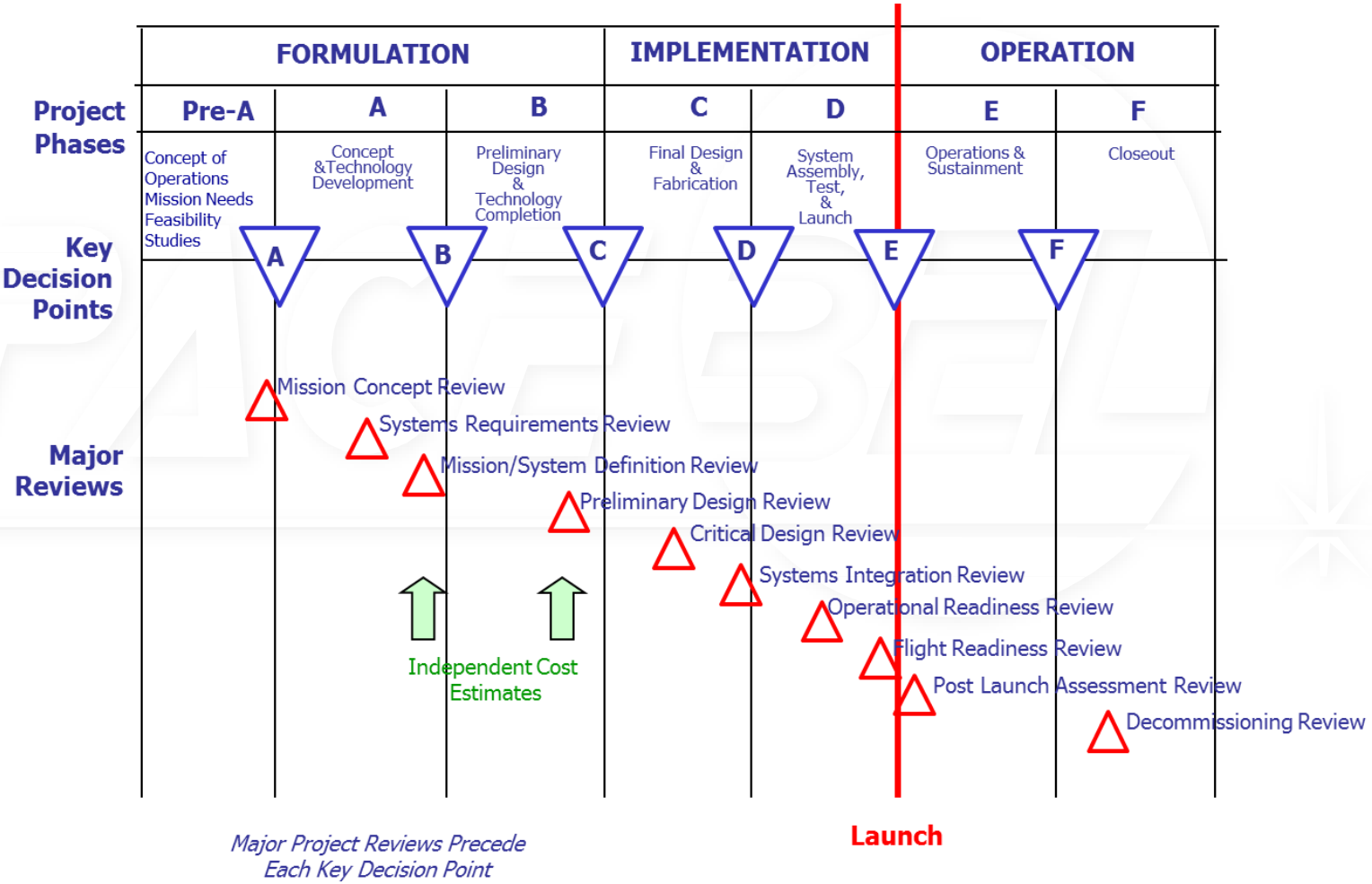
The Process

- Phases
- Lifecycle
- Verification and Validation
- Standards
- Documentation

Copyright © 2017 by SPACEBEL – All rights reserved

Phases

On Board **Software** usually comes (too) **late** in the overall **Spacecraft** development:



Copyright © 2017 by SPACEBEL – All rights reserved

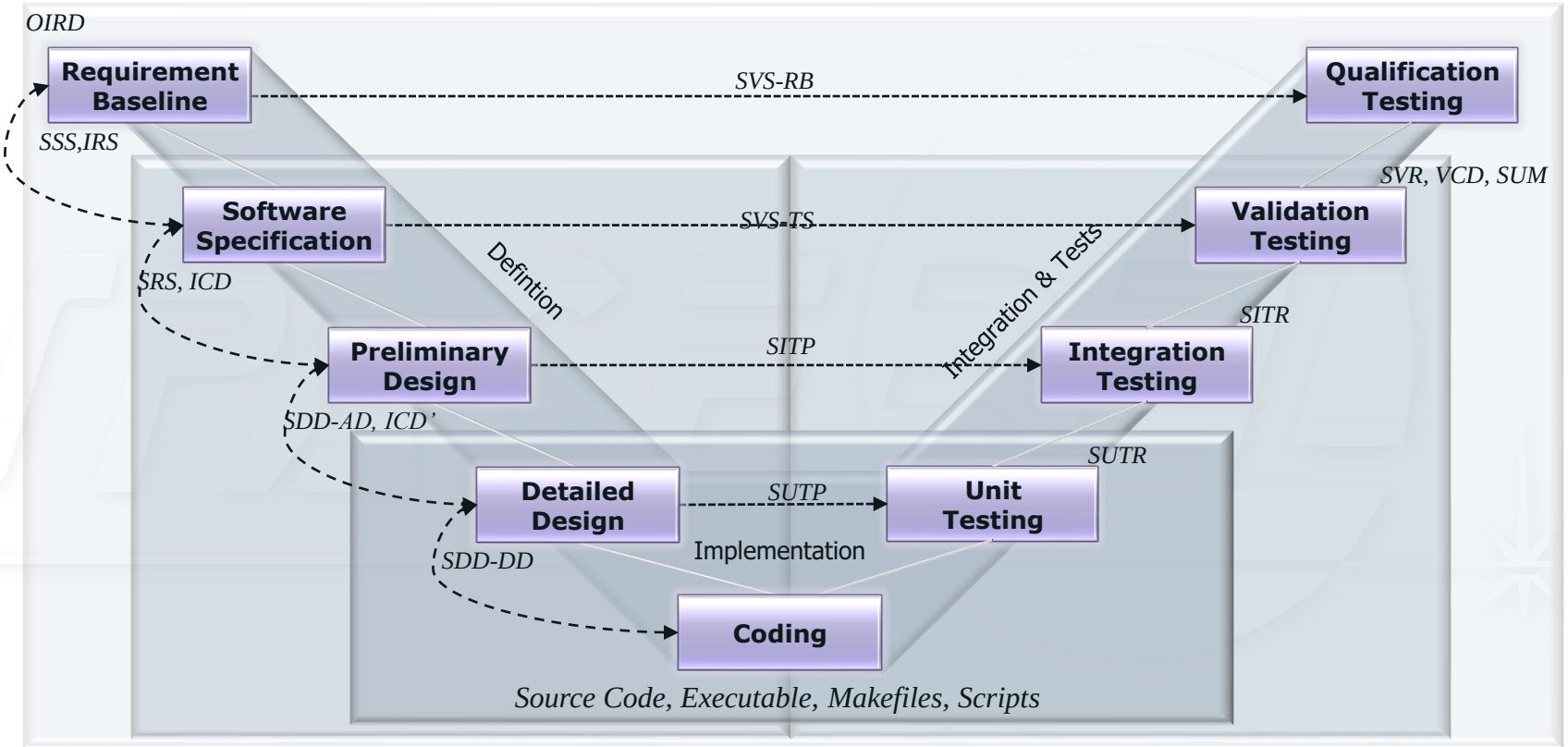
Lifecycle

On Board Software Development

V Lifecycle

RFI, RFQ, RFP, ITT, SOW

PMP, PAP, CMP, SDP, SVVP



SRR (SWRR) PDR (DDR) TRR TRB DRB CDR QR AR

Phase A **Phase B** **Phase C** **Phase D** **Phase E**

Copyright © 2017 by SPACEBEL – All rights reserved

Production

First half of the Lifecycle

Through Refinment Process

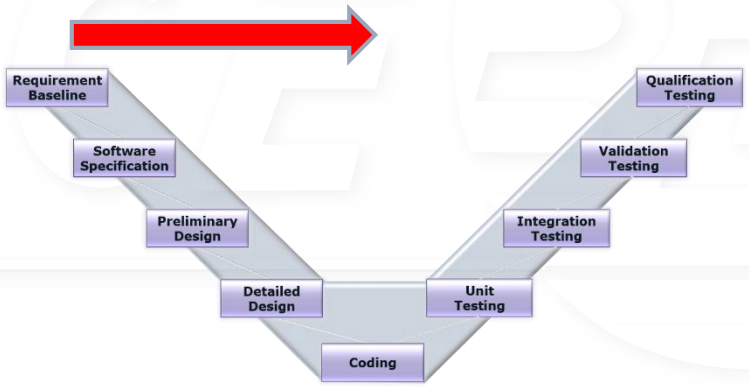
Requirements
System and mission needs and constraints

Specification
Software functionality and

Architecture
components and interfaces

Design
Algorithms and data structure

Coding
Source and object code



Supported by Development Environment
(methods and tools)

Requirement Engineering

Software Modeller

Code Generator

Copyright © 2017 by SPACEBEL – All rights reserved

Validation

Second half of the Lifecycle
After the Development

Through Tests Campaign

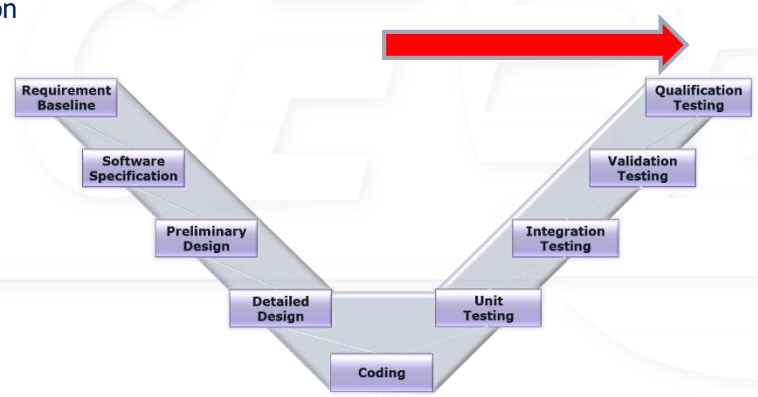
Unit Tests
against Detailed Design
On Target

Integration Tests
against Architecture and Interface Definition

Validation Tests*
against Technical Specification
on Software Vailidation Facility

Qualification Tests
against Requirement Baseline
on Hardware Model

*some requirement may alternatively be validated
by Analogy,
by Analysis of Design,
by Inspection of Code



Supported by Test Facilities

Software Test Benches*

Hardware Models

Hybrid Facilities

*also known as
Software Validation Facility

Independent Sofware Verification and Validation (required for Cat B)
must be performed by an **independent** team or company
in addition to normal Verification and Validation

Copyright © 2017 by SPACEBEL – All rights reserved

Vérification

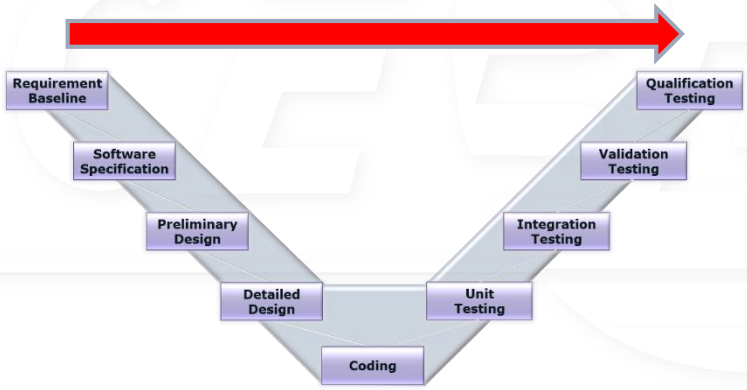
All along the lifecycle

Through Reviews and Analysis

- Peer Review
- Cross Reading
- Static Code Analysis
- Schedulability Analysis

Supported by Software Engineering Tools

- Model Verifiers
- Static Code Analyzer
- Traceability Matrices



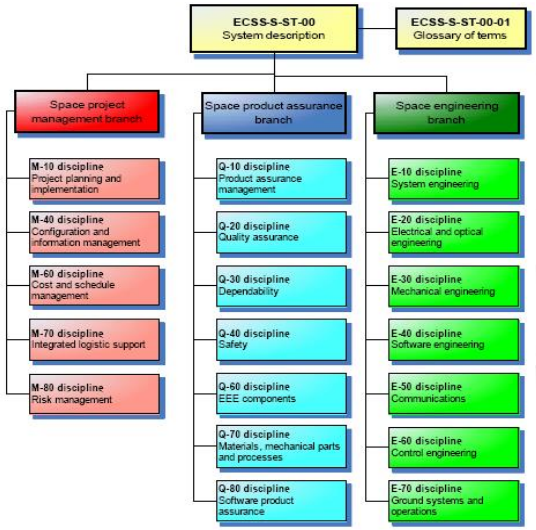
On Board Software Verification and Validation represent a very significant part of the total development effort

Independent Sofware Verification and Validation (required for Cat B)
must be performed by an **independent** team or company
in addition to normal Verification and Validation

Copyright © 2017 by SPACEBEL – All rights reserved

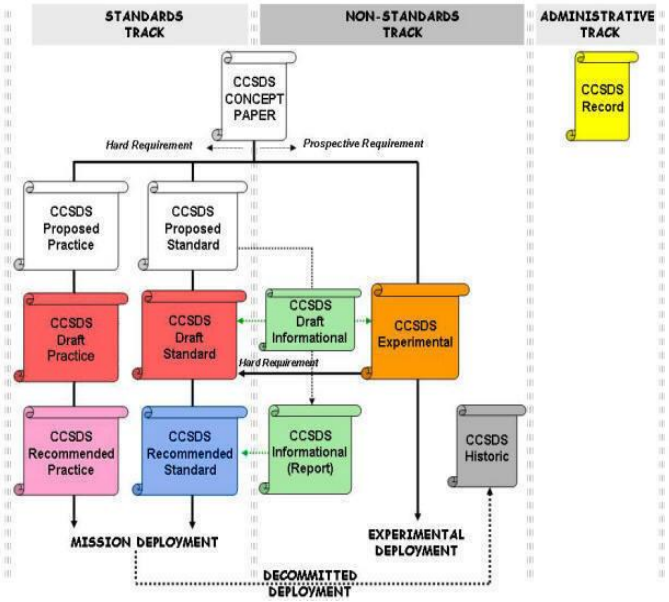
Standards

European Cooperation for Space Standardization



Standards:
- Management Standards
  → ECSS-M-XXX
- Product Assurance Standards
  → ECSS-Q-XXX
- Engineering Standards
  → ECSS-E-XXX

Consultative Committee for Space Data Systems



Reports:
- **Blue**: Recommended Standards
- **Magenta**: Recommended Practices
- **Green**: Informational Reports
- **Orange**: Experimental or ongoing research
- **Yellow**: Record, but not Historical
- **Silver**: Historical

See also D0-178-C, ANSI / IEEE, …

Copyright © 2017 by SPACEBEL – All rights reserved

Documentation

Related file	DRL item (e.g. Plan, document, file, report, form, matrix)	DRL item having a DRD	SRR	PDR	CDR	QR	AR	ORR
RB	Software system specification (SSS)		✔					
	Interface requirements document (IRD)		✔					
	Safety and dependability analysis results for lower level suppliers		✔					
TS	Software requirements specification (SRS)			✔				
	Software interface control document (ICD)			✔	✔			
DDF	Software design document (SDD)			✔	✔			
	Software configuration file (SCF)			✔	✔	✔	✔	✔
	[illegible]					✔	✔	
	Software user manual (SUM)				✔	✔	✔	
	Software source code and media labels				✔			
	Software product and media labels					✔	✔	✔
	Training material					✔		
	[illegible]			✔				
	Software validation plan (SValP)			✔				
	Independent software verification & validation plan		✔	✔				
	Software integration test plan (SUITP)			✔	✔			

MGT Management File
- Software development plan
- Software configuration management plan (DRD in ECSS-M-ST-40)
- Software review plan
- ...

RB Requirements Baseline
- Software system specification
- Software interface requirements document
- ...

DDF Design Definition File
- Software design document
- Software configuration file (DRD in ECSS-M-ST-40)
- Software release document
- Software user manual
- ...

MF Maintenance File
- Maintenance plan (without DRD)
- Migration plan (without DRD)
- ...

PAF Product Assurance File
- Software product assurance plan
- Software product assurance milestone report
- Software product assurance requirements for suppliers
- ...

TS Technical Specification
- Software requirements specification
- Interface control document
- ...

DJF Design Justification File
- Software validation plan
- Software verification plan
- Software unit and integration plan
- Software validation specification
- Software reuse file
- Software verification report
- ...

OP Operational
- Operational plan
- Operational testing specification

Copyright © 2017 by SPACEBEL – All rights reserved

The Architectures

- Functional Architecture
- Static Architecture
- Dynamic Architecture
- Deployment Architecture

Copyright © 2017 by SPACEBEL – All rights reserved

Functional Architecture: Functions

Note: interactions between functions are not depicted

TM/TC *Telemetry & Telecommand*

space/ground communications or communications between spacecraft

HK *Housekeeping*

Gathering, filtering and reporting of on board acquired data

MON *Monitoring*

detection of on board events based on ranges or thresholds or trends

FDIR *Fault Detection Isolation and Recovery*

for the on board software and system dependability

MMGT *Mission Management*

for the execution of the mission timeline

SMGT *Spacecraft Modes Management*

handles the on-board system through the different mission phases
defines the level of autonomy,

GNC/AOCS/ACNS *Attitude and Orbit Control Software*

Computes the actuators commands from the sensors measurement in order to control the spacecraft attitude and position.
Control the propulsion system.

THERM *Thermal Management*

for the temperature control of the spacecraft e.g. through thermal heater lines,

PWR *Electrical Power Supply Management*

distributes the power coming from the battery and the solar arrays
and manages battery charge / discharge,

RDP/HRM *Release, Deployment and Pyrotechnic Activation Management*

for solar arrays/antenna deployment,
for propulsion arming sequence before firing, or
for specific needs like shield jettison or spacecrafts composites dispatching,

OBT *On-Board Time Management*

For the synchronization of the on-board clock with ground time,

ACQ *Data Acquisition*

Acquisition of on board data
e.g. according to polling sequence table
and depending on mode

SADM *Solar Array Drive Mechanisms Management*

for the optimal alignment of a satellite's solar panels towards the Sun.

SM *Storage Management*

for the storage of TM in case of Earth link unavailability

EQPT *Equipment Management*

for the maintain of the equipment table as the reflect of the actual equipment status, and the management of equipment interface,

RF *Radio Frequency Management*

for the communications with ground or with other sapcecrafts
e.g. through S-Band or X-Band links.

PL *Payload Management*

for scientific payload or additional units

(very mission specific)

CM *Context Management*

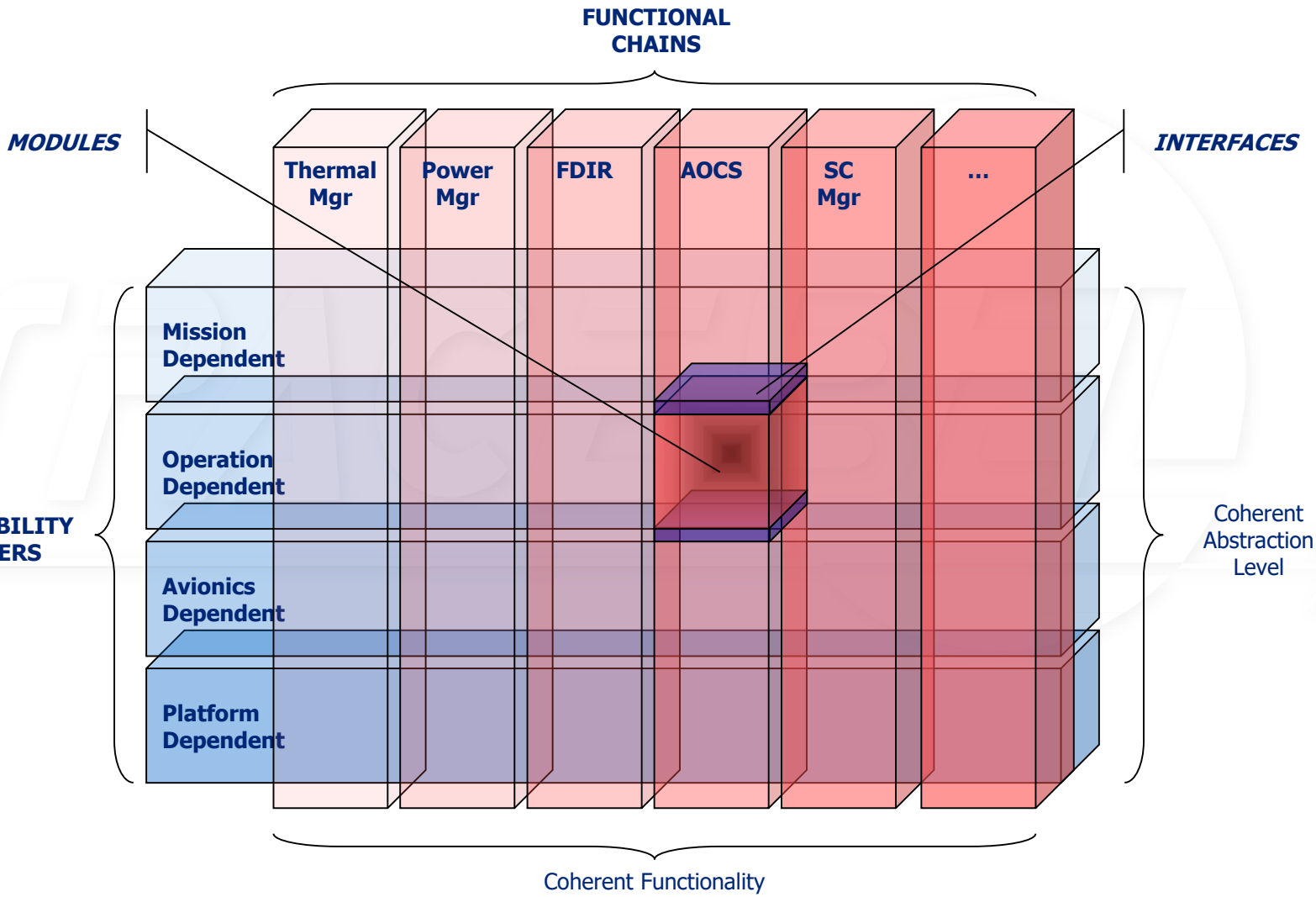
Saves the context (failure history buffer, GNC states, OBSW patches, equipment table, mission timeline) in case of processor failure.

ETC *… And Many Others*

And many other functions depending on the platform bus and on the mission

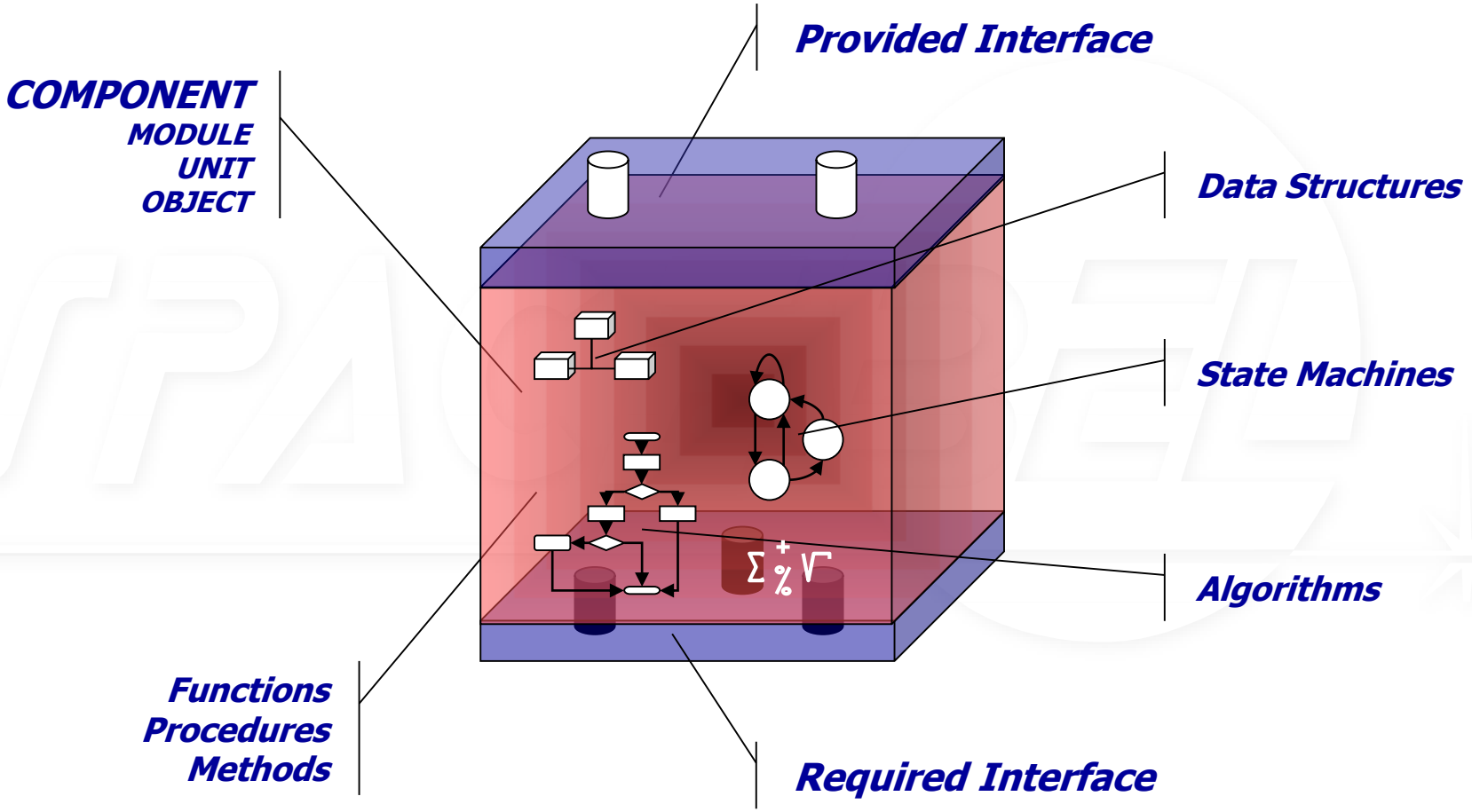
Copyright © 2017 by SPACEBEL – All rights reserved

Static Architecture (2/4): Structure



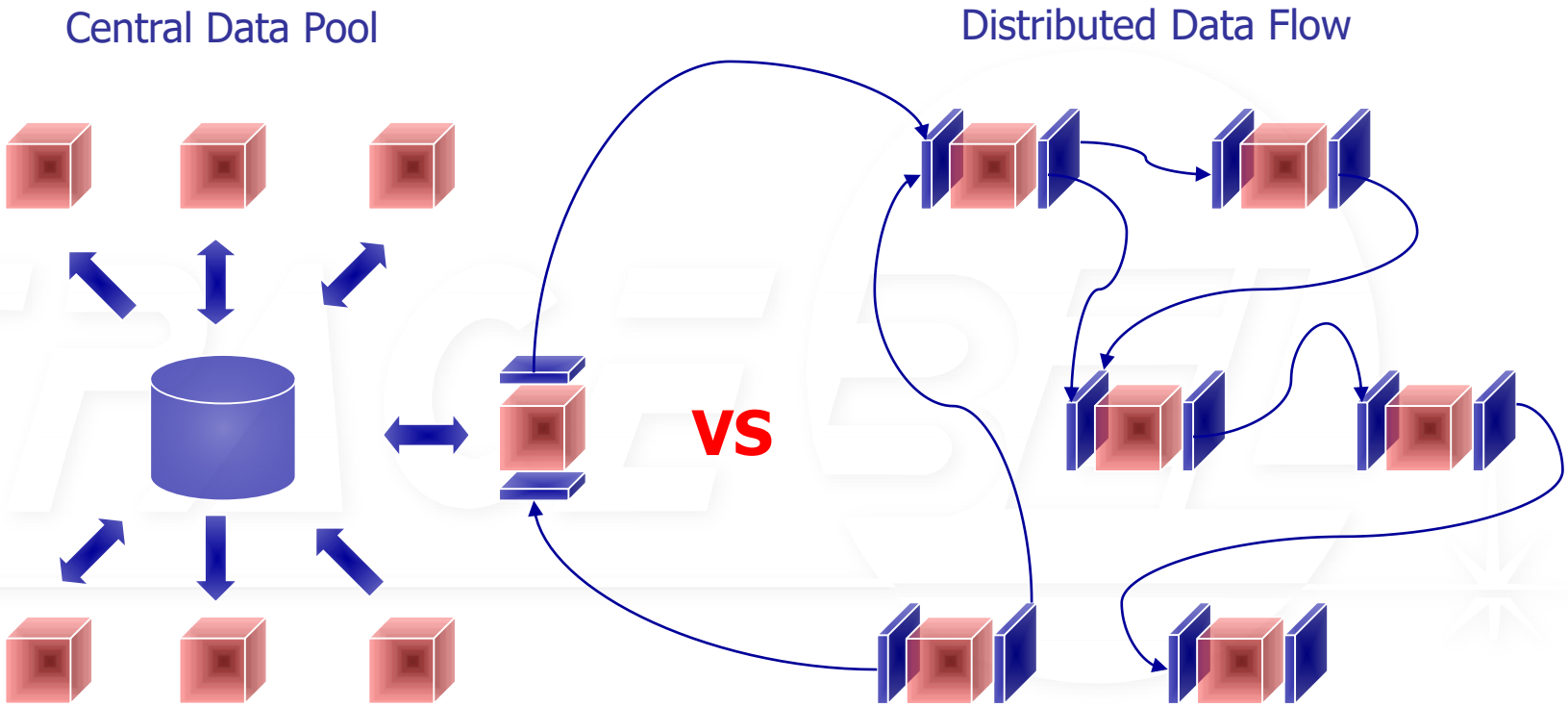
Copyright © 2017 by SPACEBEL – All rights reserved

Static Architecture (3/4): Components



Copyright © 2017 by SPACEBEL – All rights reserved

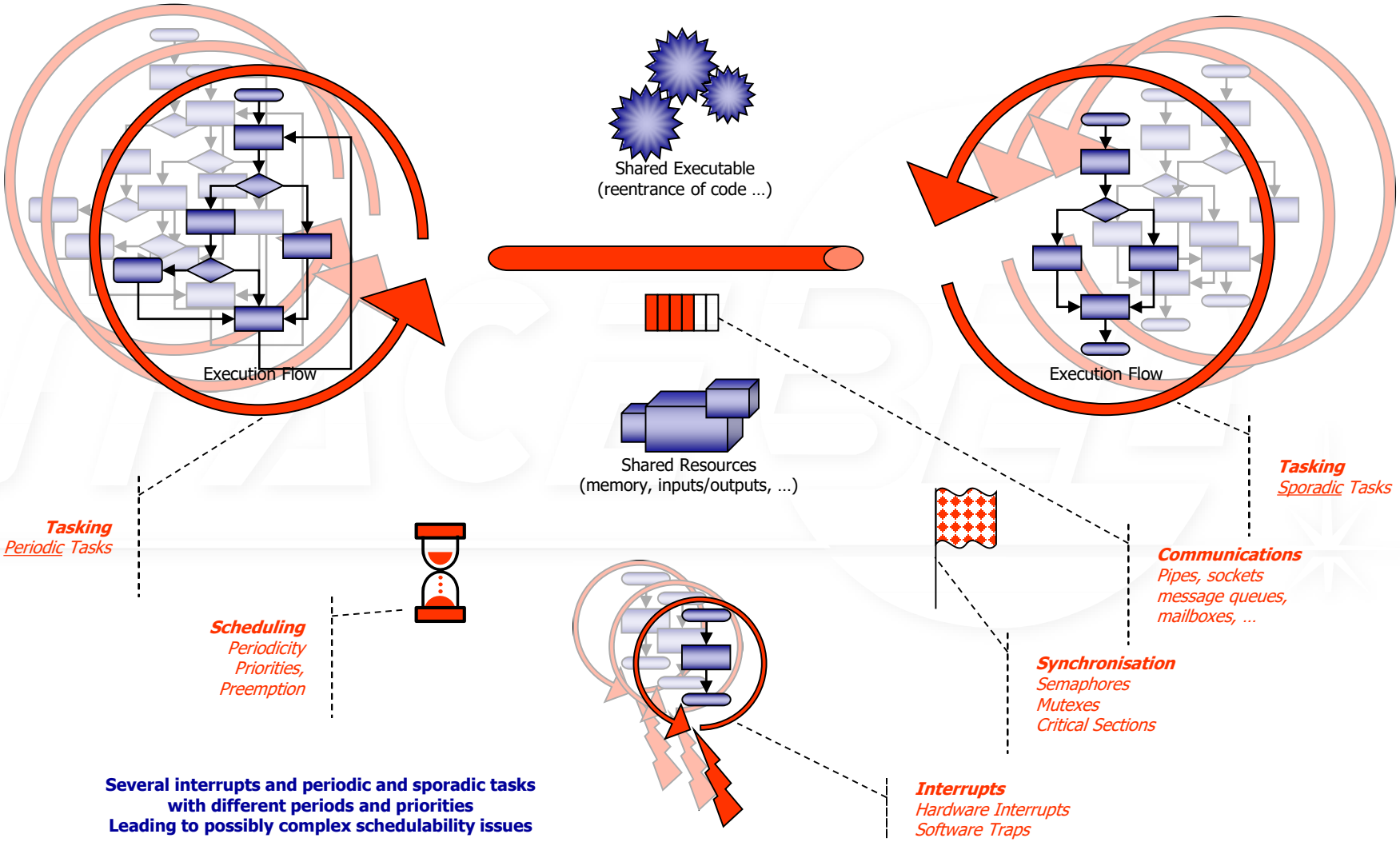
Static Architecture (4/4): Paradigms



Impact on Deployment over Multi Processors, Multi Cores, Multi Partition
(see Deployment Architecture further down)

Copyright © 2017 by SPACEBEL – All rights reserved

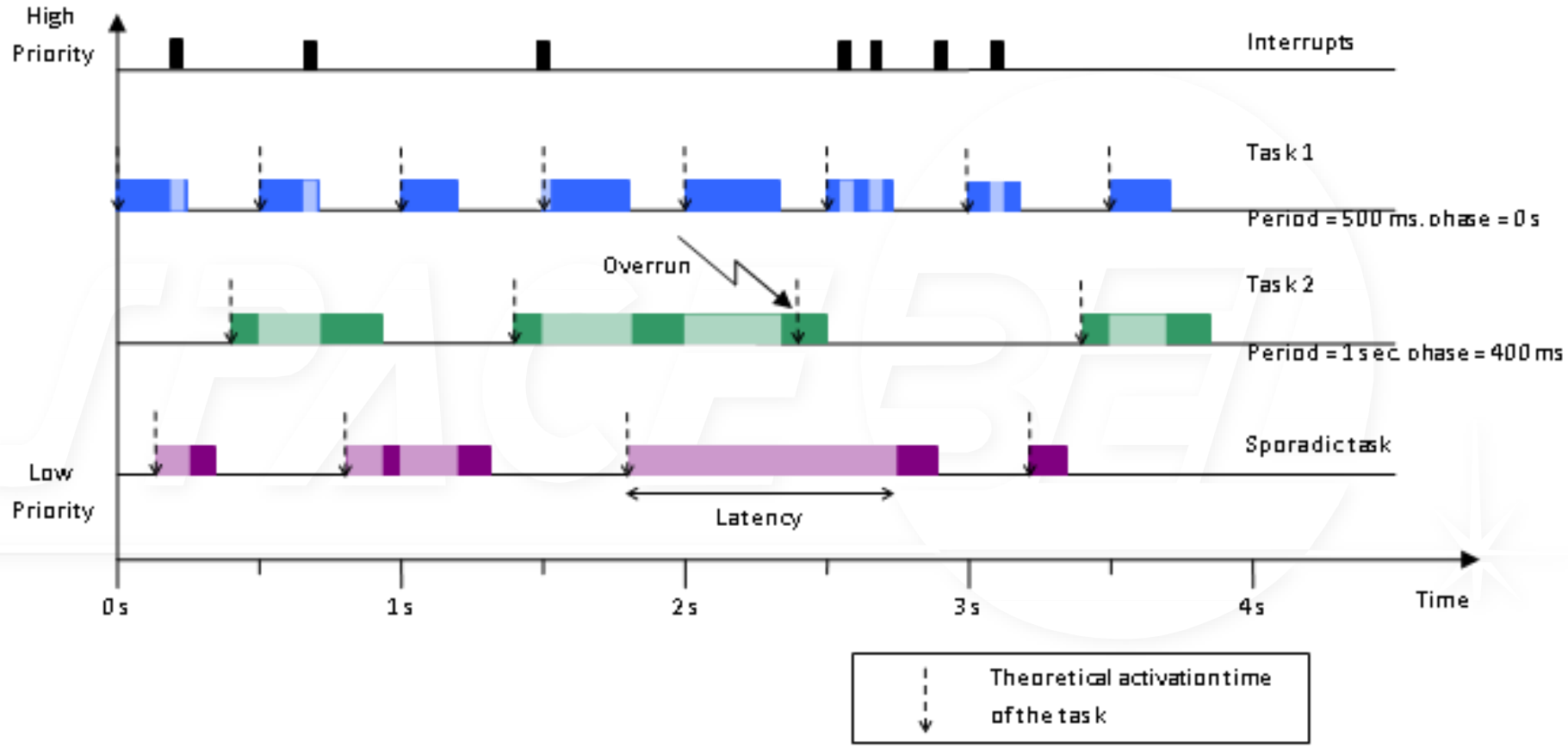
Dynamic Architecture (2/6) : Concepts



Several interrupts and periodic and sporadic tasks with different periods and priorities Leading to possibly complex schedulability issues

Copyright © 2017 by SPACEBEL – All rights reserved

Dynamic Architecture (3/6) : Example



Copyright © 2017 by SPACEBEL – All rights reserved

Dynamic Architecture (4/6): Computational Model

- Computational Models
  - RMA: Rate Monotonic Algorithm
    Static priority preemptive scheduling. Applies to cyclic jobs. Shorter cycle get higher priority
  - DMA: Deadline Monotonic Algorithm
    Static priority preemptive scheduling. Applies also to sporadic jobs. Shorter deadline get higher priority
  - EDF: Earliest Deadline First Algorithm
    Dynamic priority preemptive scheduling. process closest to its deadline get highest priority
  - RCM: Ravenscar Computational Model
    Fixed-priority preemptive system with tight restrictions on tasking and synchronisation such as priority ceiling that optimally bound priority inversion, to achieve lock-free mutual exclusion and to avoid deadlocks. Warrants static analysability of the source code and predictability of execution

- *TSP: Time and Space Partitioning*
  hierarchical superposition of two computational models :
  round robin scheduling of partitions and
  fixed priority scheduling of tasks within partitions

See also Schedulability Analysis

Copyright © 2017 by SPACEBEL – All rights reserved

Dynamic Architecture (5/6): Schedulability

- Proof of software schedulability (determism)
  - Dynamic Testing at Run Time (how to prove exhaustivity?)
  - Static Analysis (superior to testing)
    - Selected Computational Model (see dynamic architecture)
      - Implementation is assumed to comply to model
      - Mathematical schedulability criteria
    - Software Budget Report (estimated – a priori - or measured – a posteriori)
      - Processor Utilization
      - Worst Case Execution Times (WCET)
      - (See also Memory Footprint and Stack Usage)

Copyright © 2017 by SPACEBEL – All rights reserved

Dynamic Architecture (6/6): Schedulability

- Rate Monotonic Scheduling
- Deadline Monotonic Scheduling
- Earliest Deadlin First
- Ravenscar Profile

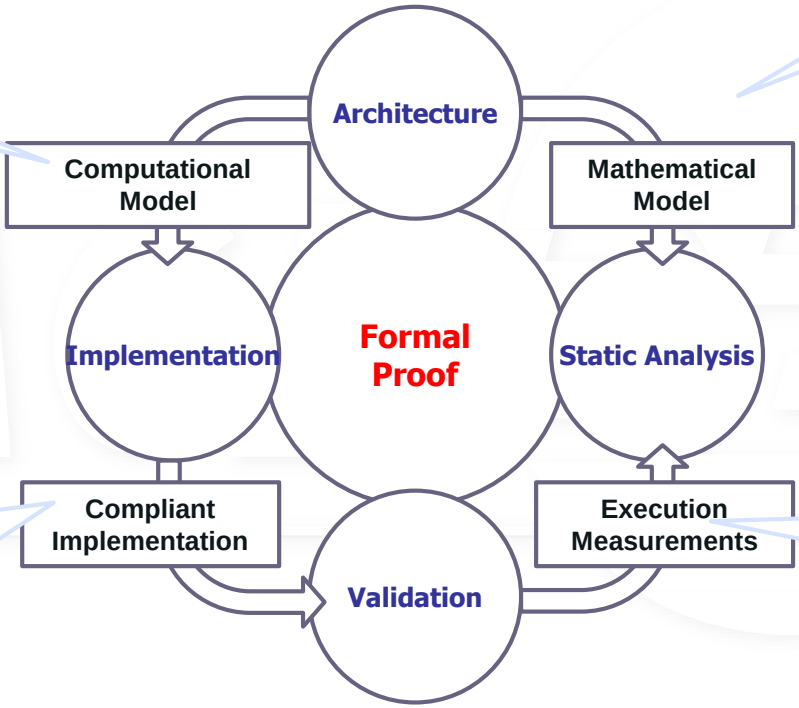
(See Dynamic Architectures)

Schedulability Condition

(Rate Monotonic Only !)

$$U = \sum_{i=1}^{n} C_i/T_i \leq n(\sqrt[n]{2} - 1)$$

$$\lim_{n \to \infty} n(\sqrt[n]{2} - 1) = \ln 2 \approx 0.693147\ldots$$



- Programming Model
- Design and Coding Rules
- Code Generation

Processor Utilization
Worst Case Execution Time

(Based on realistic operational scenarios)

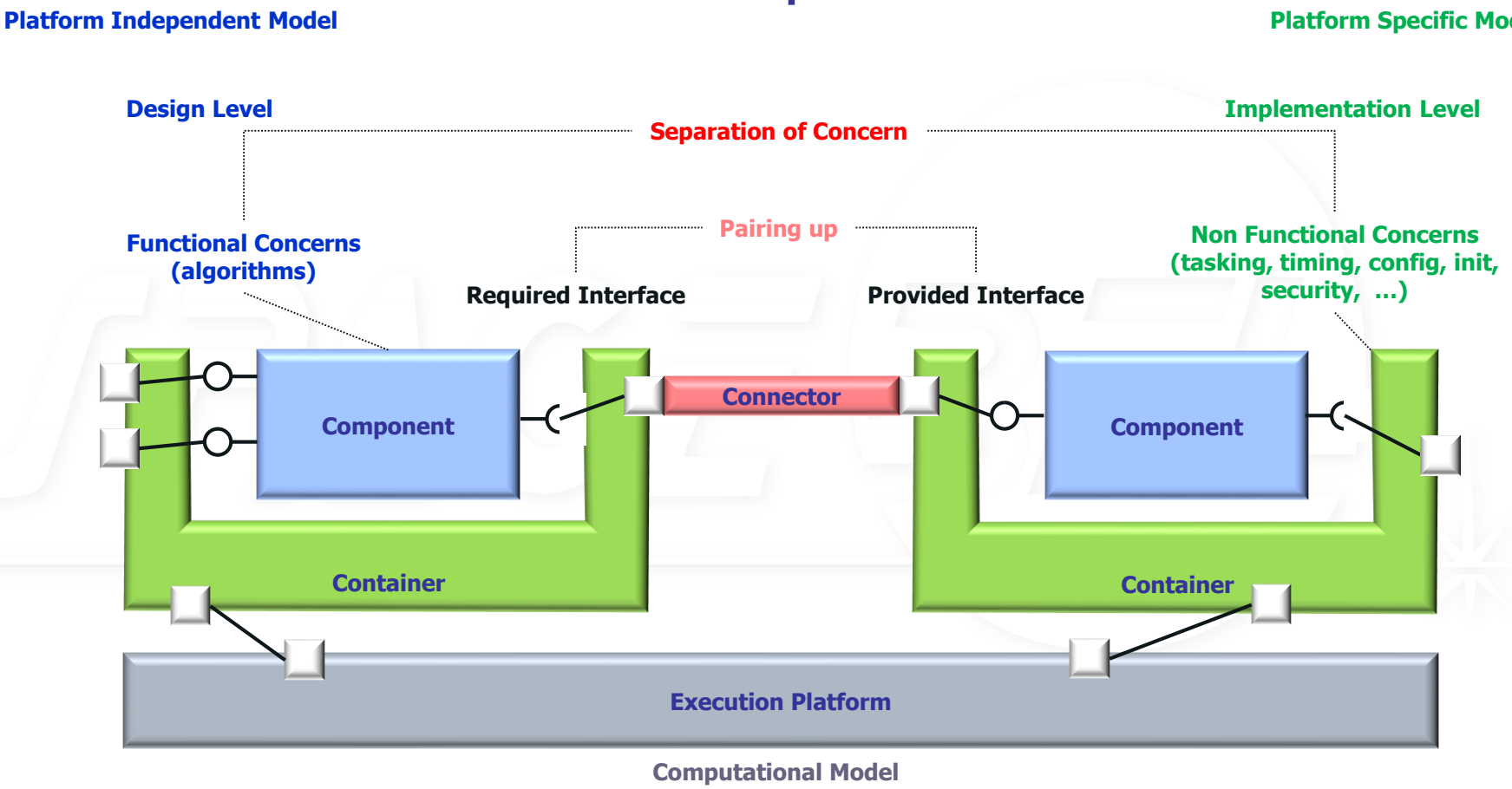
Compliance to a selected Computational Model allows for Static Analysis and Formal Check of Schedulability Conditions, based on corresponding Mathematical Model fed by actual Measurement from execution of realistic scenarios.

To this respect, Static analysis is superior to testing, which faces exhaustivity issue in real-scale systems.

Copyright © 2017 by SPACEBEL – All rights reserved

Deployment Architecture (3/3): SCM

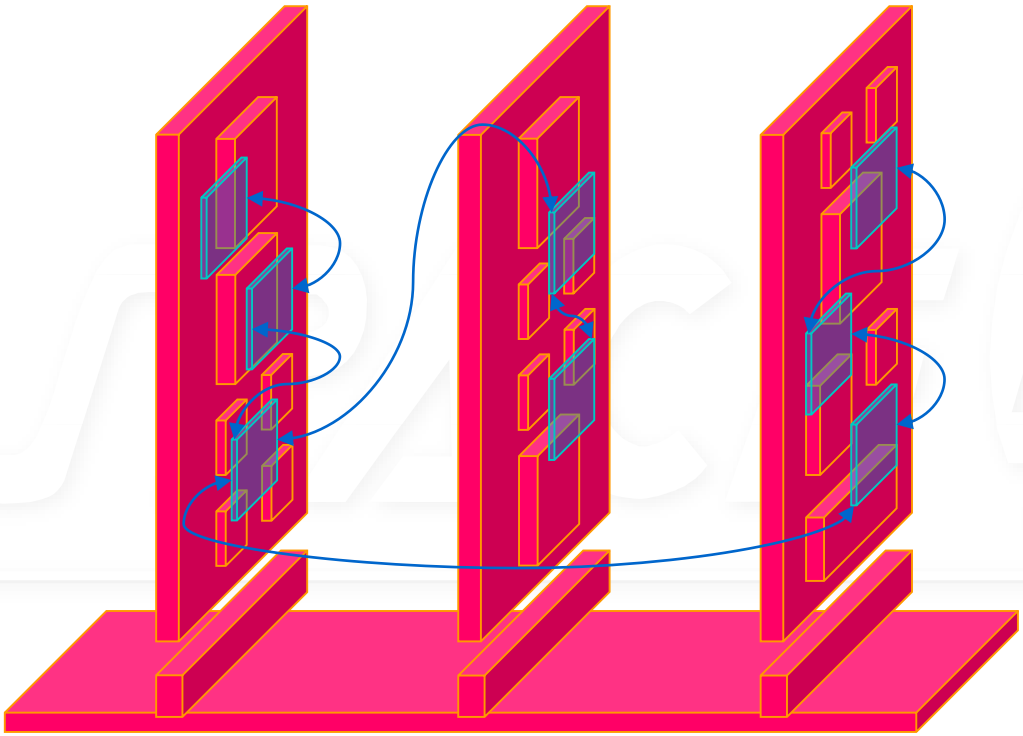
Software Component Model



Composition **Reusability** **Replaceability**

Copyright © 2017 by SPACEBEL – All rights reserved

Deployment Architecture (2/3): Mapping



A deployment architecture depicts the **mapping**

of a **logical architecture**
(software components)

to a **physical environment**
(hardware resources).

The physical environment includes the computing nodes, processors, memory, storage devices, and other hardware and communication devices

Copyright © 2017 by SPACEBEL – All rights reserved

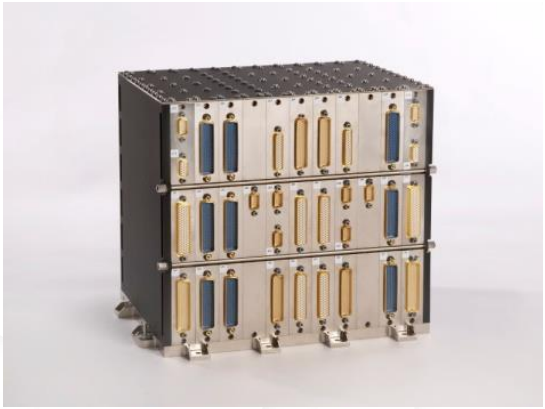
The Environments

- Cross Development
  - Development Environment
  - Validation Environment
  - Execution Environment

Copyright © 2017 by SPACEBEL – All rights reserved

Environments: Cross Environment

Cross
Development
Environment



Development
Environment:

≠

Execution
Environment:

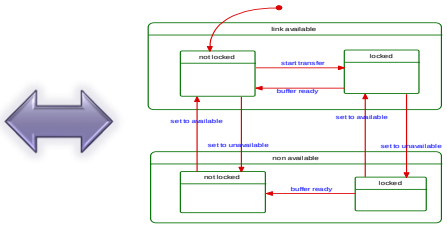
Powerfull Development Workstations
Confortable Operating System
User Friendly Production Tools

On Board Computer with Limited Resources
Real Time Operating System with Constraints
Embedded Executable Software Image

Copyright © 2017 by SPACEBEL – All rights reserved

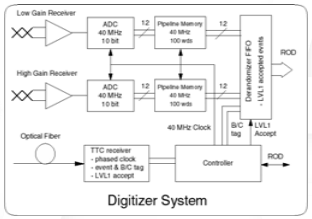
Development Environment: Modeling

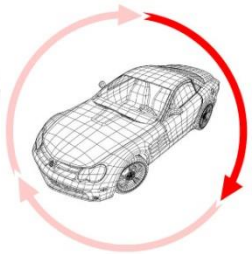

```
/** Note: This function is only called from the event manager sporadic activities.
 *  Extraction of the event data is performed within EVTR_execute_sporadic_activities. */
TM_DECLARE_NEW_MESSAGE (TM_room, TM_ptr, TM_FIXED_SIZE + EVT_MAX_EVENT_DATA_LENGTH)

uint32                  data_size;
uint8 *                 data_ptr = TM_get_data(TM_ptr);

/** Verify event_id input. It must be less than EVT_NB_EVENTS. Note that this only protects from table overflow:
 *  This should indeed never happen as the data has been checked when it was put in the queue.
 *  Therefore only return to avoid overflowing, no report is generated. */
if (event_id >= EVT_NB_EVENTS)
{
  return;
}

/** The event must be enabled for reporting: If the event id is disabled, return without generating a report. */
if (EVTR_config[event_id].reported == False)
{
  return;
}

/** Event data size is limited by the maximum event length. If the generation of this report would create
 *  a telemetry packet with data size larger than EVT_MAX_EVENT_DATA_LENGTH, truncate to EVT_MAX_EVENT_DATA_LENGTH. */
data_size += sizeof(EVT_RID_t) + event_data_length;
if (data_size > EVT_MAX_EVENT_DATA_LENGTH)
{
  data_size = EVT_MAX_EVENT_DATA_LENGTH;
}

GU_memcpy( (data_ptr , &event_id , sizeof(EVT_RID_t));
data_ptr += sizeof(EVT_RID_t);
if (event_data != NULLPTR)
{
  GU_memcpy(data_ptr, event_data, data_size - sizeof(EVT_RID_t));
}

/** Generate the event by creating the corresponding service 5 packet. The PUS subtype (between 1 and 4)
 *  is obtained from the EVTR_config configuration table. */
TM_create(TM_room,
          TMTC_ERS,
          TMTC_ERS,
          EVTR_config[event_id].event_subtype,
          0, /* data are already at the correct place */
          data_size,
          ROUTE_GROUND_ID);

TM_send_telemetry(TM_ptr,
                  data_size + TM_FIXED_SIZE,
                  True,
                  False);
```

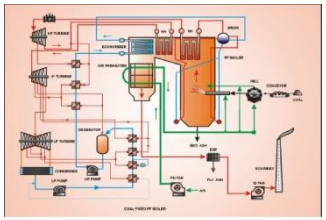


Software has the rare property that it allows us to directly evolve models into fully-fledged implementations without changing the engineering medium, tools, or methods

The software model may evolve into the system it was modeling in a seamless process.



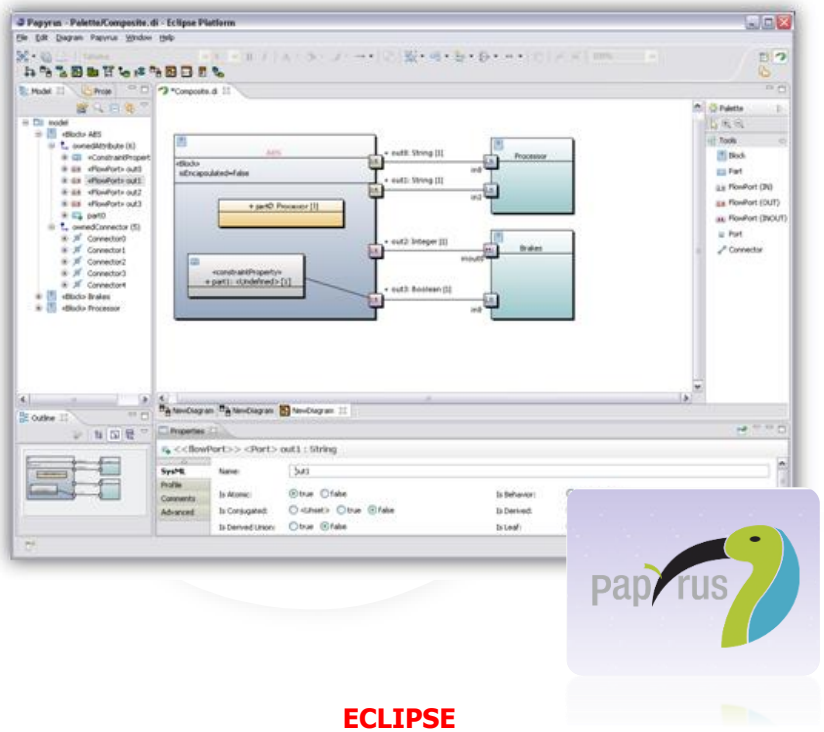




Copyright © 2017 by SPACEBEL – All rights reserved

Development Environment: Modeling

- Paradigm Shift
  - From Programming to Modelling
  - Model Based Software Engineering
- Modeling Domains
  - Requirements
  - Architecture Modeling,
  - Data Modeling,
  - Behaviour Modeling
- Modelling Languages
  - UML, SYSML, AADL, AAML, SDL, …
- Model Verification
  - Strong Syntax, Formal Verification
- Automatic Generation
  - Code
  - Documents



ECLIPSE
Integrated Development Environment (IDE) and Eclipse Modelling Frameworl (EMF)

Copyright © 2017 by SPACEBEL – All rights reserved

Development Environment: Programming

- Languages
  - C:
    procedural language, widely used, poor expressivness but good control, well suited to system programming, efficient code but error prone
  - Ada:
    strong typing, well suited to embedded and real-time programming, mainly used in launchers
  - C++ :
    object oriented, based on C, less efficient due to object orientation, not used or poorly used on board so far
  - Java :
    object oriented, interpreted language, under investigation, possibly for OBCP
  - Assembler
    low level language, for specific usage

Main programming paradigms are
Procedural programming vs
Object Oriented programming (not used so far in on board software)

Copyright © 2017 by SPACEBEL – All rights reserved

Development Environment: Generating

- Cross Environment (remember – see above)
  - → Cross Compiler

Copyright © 2017 by SPACEBEL – All rights reserved

Development Environment: Compiling

- Cross Environment (remember – see above)
  - → Cross Compiler

Compiling
is the process of translating computer code written in one programming language (the source language) into another language (the target language).

Copyright © 2017 by SPACEBEL – All rights reserved

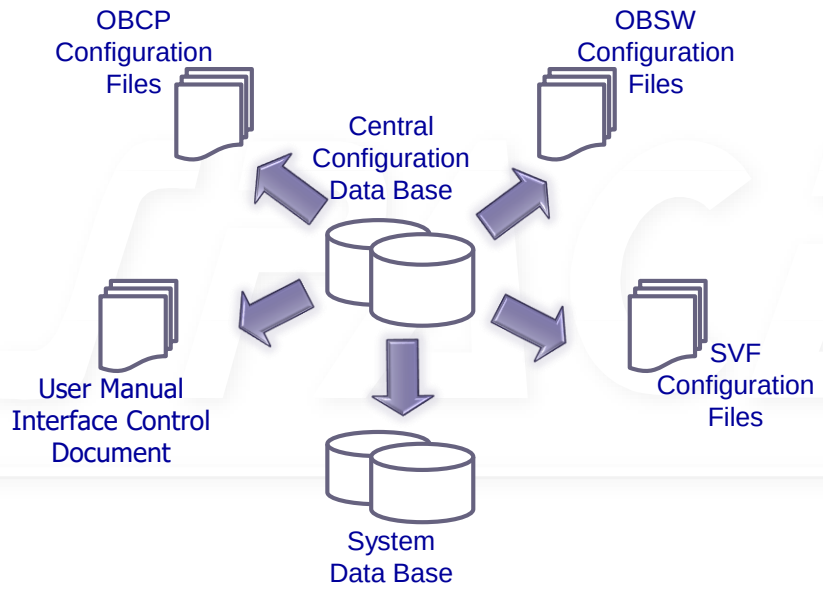
Development Environment: Debugging

- Cross Environment (remember – see above)
  - → Remote Debugging
  - → Debug Support Unit

Debugging
is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system (wikipedia)

Copyright © 2017 by SPACEBEL – All rights reserved

Development Environment: Production



Centralised Database

- Contains …
  On-board software configuration,
  Constant values, and
  Parameters useful for ground

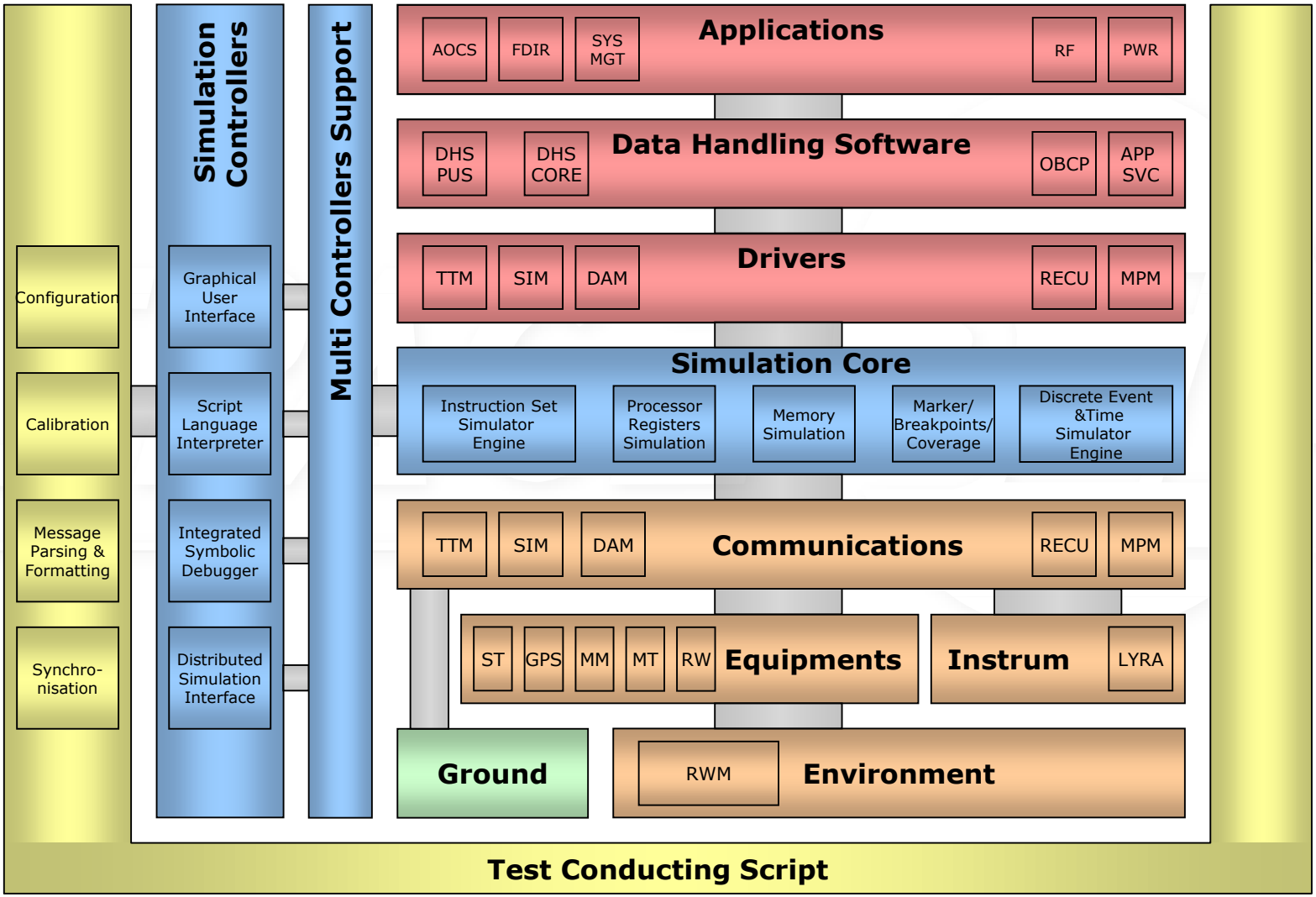
  tasks, message queues, observable parameters, events, actions associated with events, default housekeeping, default monitoring, patch areas, Drivers commands, TC function ID's (PUS 8, 1), Memory partitions, File partitions, On-board stores, Default on-board storage allocation, PUS services and sub-services, Types of the parameters in the PUS TC/TM, Error codes, Application ID …
- Ensure coherency between …
  the different outputs

Production Tools

- Support Automatic generation of …
  Configuration Files and
  Documentation
- Allow for easy configuration

Copyright © 2017 by SPACEBEL – All rights reserved

Validation Environment: Testing



OBSW

Generic SVF Component

Platform dependent SVF Component

Test Conducting

Copyright © 2017 by SPACEBEL – All rights reserved

Execution Environment: µP

- Hardened Space Qualified processors
  1990: 1750 – 16 Bits – 2 MIPS
  2000 : ERC32 – SPARC V7 - 32 Bits – 20 Mhz – 14 MIPS
  2010 : LEON2 – SPARC V8 - 32 Bits – Cache – 5 Stages Pipeline - 100 Mhz – 84 MIPS
  2105 : LEON3 – Dual Core LEON – 7 Stages Pipeline - MMU -100 Mhz – 84 MIPS
  2105 : LEON4 – Quad Core LEON

- Commercial of the Shelf
  - ARM
  - Power PC (1600 MIPS but sensitive to SEEs)

- Coming
  - RISC V

See Avionics Overview

Copyright © 2017 by SPACEBEL – All rights reserved

Execution Environment: RTOS

- RTOS: Real Time Operating Systems
  - VxWorks (Commercial)
  - RTEMS (Open Source)
  - Linux RT (Not widely used … in OBSW)
  - FreeRTOS
  - …
- TSP: Time and Space Partitionning
  - Hypervisor
  - µKernel

Copyright © 2017 by SPACEBEL – All rights reserved

The Trends

- Hardware Software Co Engineering
- Cyber Security
- Artificial Intelligence

Copyright © 2017 by SPACEBEL – All rights reserved

File Based Operation

- Files and file systems
  Also On Board spacecrafts
  - Applications
    - Store Housekeeping and Science Data in Files and File Systems
      - And Download to ground
    - Process Command from Files
      - And Upload from Ground
  - Techniques
    generic approach instead of ad hoc mechanisms
    - File Aware Services
    - File System on Board
    - File Transfer Protocols

Xilinx - Zynq – Zybo – ARM+FPGA

Copyright © 2017 by SPACEBEL – All rights reserved

HW SW Co Engineering

- Field Programmable Gate Arrays (FPGA), System on a Chip (SoC) …
  Also On Board spacecrafts
  - Applications
    - µP in FPGA
      e.g. Prototyping
    - Heavy Processing Algorithms
      e.g. Image Processing, Feature Extraction, Vision Based Navigation
  - Techniques
    - Programmable HW (Is this HW or SW?)
    - Need Space Hardened FPGA
    - Specific Language VHDL (but possible translation from C)
    - Specific Development Environment (but increasingly combined)

Xilinx - Zynq – Zybo – ARM+FPGA

Copyright © 2017 by SPACEBEL – All rights reserved

Cyber Security

- General Concern,
  also in Space Business,
  also On Board Spacecraft
  - Applications
    - Space Communication Security
      - Telecommands Authentication
        (to protect at the minimum the commanding link to the spacecraft)
      - Telemetry Encryption
        (to protect mission products from unauthorized access)
  - Techniques
    - Cryptographic and Key Management algorithms and protocols
    - Specific Hardware Support

Copyright © 2017 by SPACEBEL – All rights reserved

Artificial Intelligence

- Global Trend
  also in Space Business,
  also On Board Spacecraft

  - Applications
    - On Board Decision: Mission Planning
    - On Board Detection: Faults, Image Processing.
      Cloud Detection, Fire Detection …

  - Techniques
    - Big Data (Volume, Velocity and Variety, Veracity and Value)
    - Machine Learning / Deep Learning (supervised, unsupervised, semi supervised, reinforcement)
      set of techniques that allow computers to learn from data without being explicitly programmed
    - VPU or GPU support

Copyright © 2017 by SPACEBEL – All rights reserved