Spacecraft
On Board Software

December 2017
• OBSW Characteristics
• OBSW Process
• OBSW Architecture
• OBSW Environments
● OBSW Characteristics
● OBSW Process
● OBSW Architecture
● OBSW Environments
- OBSW Characteristics
  - OBSW Constraints
  - OBSW Dependability
  - OBSW Criticality
  - OBSW Properties
- OBSW Processes
- OBSW Architectures
- OBSW Environments
OBSW: Constraints

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

- **Embedded** Software
  -> Cross Development Environment, bounded Memory Footprint and Processor Consumption

- **On Board** Software
  -> Single Event Effects (Upset, Latch Up), Memory Scrubbing

- **Real Time** Software
  -> Processor load, scheduling issues, deadlines

- **Deterministic** Software
  -> No dynamic thread creation or memory allocation, budget and schedulability

- **Remote** Software
  -> Need for autonomy

- **Critical** Software (see further down)
  -> Possible catastrophic consequences of failure

- **Dependable** Software (see further down)
  -> Need for high Reliability, Availability, Maintainability and Safety
● OBSW Characteristics
  ● OBSW Constraints
  ● **OBSW Dependability**
  ● OBSW Criticality
  ● OBSW Properties
● OBSW Process
● OBSW Architectures
● OBSW Environments
OBSW: Dependability

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

**Integrity**
- Maintenance of data consistency

**Security**
- Non disclosure of unauthorized info

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

**Analysis**
- **SCA:** Software Criticality Analysis (see next slide)
- **HSIA:** Hardware Software Interaction Analysis
- **FMECA:** Failure Mode Effects and Criticality Analysis
- **FTA:** Fault Tree Analysis
- **FHA:** Functional Hazard Analysis
- **SCCFA:** Software Common Causes Failure Analysis

**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

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

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

FDIR is imperative to guarantee a dependable and autonomous system with a minimal risk of ruinous failure.
- OBSW Characteristics
  - OBSW Constraints
  - OBSW Dependability
- **OBSW Criticality**
  - OBSW Properties
- OBSW Process
- OBSW Architectures
- OBSW Environments
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).
● OBSW Characteristics
  ● OBSW Constraints
  ● OBSW Dependability
  ● OBSW Criticality
  ● OBSW Properties
● OBSW Process
● OBSW Architectures
● OBSW Environments
OBSW: Properties

- System-software co-engineering:
  Software is part of the System.
  Software properties must be derived from System properties.

- End-to-end system response time will result into a
  → software schedulability property.

- System availability property will result into
  → software FDIR mechanisms that must have a particular behaviour.

- System performance property may result in a
  → software numerical accuracy property.
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
● OBSW Environments
● OBSW Characteristics
● OBSW Process
  ● OBSW Phases
  ● OBSW Lifecycle
  ● OBSW Verification
  ● OBSW Standards
  ● OBSW Documentation
● OBSW Architectures
● OBSW Environments
On Board Software usually comes (too) late in the overall spacecraft development:

<table>
<thead>
<tr>
<th>Project Phases</th>
<th>FORMULATION</th>
<th>IMPLEMENTATION</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pre-A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>Key Decision Points</td>
<td>Mission Concept</td>
<td>Systems Requirements</td>
<td>Final Design &amp; Fabrication</td>
</tr>
<tr>
<td>Major Reviews</td>
<td>Mission Needs</td>
<td>Mission/System Definition</td>
<td>System Assembly, Test, &amp; Launch</td>
</tr>
<tr>
<td></td>
<td>Feasibility Studies</td>
<td>Preliminary Design &amp; Technology Completion</td>
<td>Operations &amp; Sustainment</td>
</tr>
<tr>
<td></td>
<td>Concept of Operations</td>
<td>Preliminary Design &amp; Technology Completion</td>
<td>Closeout</td>
</tr>
</tbody>
</table>

Major Project Reviews Precede Each Key Decision Point

Launch
- OBSW Characteristics
- OBSW Process
  - OBSW Phases
  - **OBSW Lifecycle**
  - OBSW Verification
  - OBSW Standards
  - OBSW Documentation
- OBSW Architectures
- OBSW Environments
OBSW: Lifecycle

On Board Software development follows a formal Lifecycle

Phase A
- Requirement Baseline
  - RFI, RFQ, RFP, ITT, SOW
  - PMP, PAP, CMP, SDP, SVVP
- Software Specification
  - SSS, IRS, OCD
- Preliminary Design
  - SRR, SRS, ICD
- Detailed Design
  - PDR, SDD-AD, ICD'
- Coding
  - Source Code, Executable, Makefiles, Scripts

Phase B
- ATP
  - CDR
  - (DRB)
- VTP
  - (TRB)
- ITP
  - ATP
- UTP
  - ATP

Phase C
- ATP
  - CDR
  - (DRB)
- VTP
  - (TRB)
- ITP
  - ATP
- UTP
  - ATP

Phase D
- Qualification Test
  - VTR, VCD
  - (DRB)
- Validation Test
  - (TRB)
- Integration Test
  - (TRR)
- Unit Test
  - (TRR)

Phase E
- AR
  - PMP, PAP, CMP, SDP, SVVP
- RFI, RFQ, RFP, ITT, SOW
● OBSW Characteristics
● OBSW Process
  ● OBSW Phases
  ● OBSW Lifecycle
  ● **OBSW Verification**
  ● OBSW Standards
  ● OBSW Documentation
● OBSW Architectures
● OBSW Environments
OBSW: Verification & Validation

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

**Verification**

*All Along the Development*

*Through Reviews and Analysis*

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

**Validation**

*At the end of the Development*

*Through Tests Campaign*

- *(Unit Tests)* against Detailed Design
- *(Integration Tests)* against Architecture and Interface Definition
- Validation Tests* against Technical Specification
- Qualification Tests against Requirement Baseline

Supported by **Software Engineering Tools**

- Model Verifiers
- Static Code Analyzer
- Traceability Matrices

Supported by **Test Facilities**

- Software Test Benches
- Hardware Models
- Hybrid Facilities

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

Independent Software Verification and Validation is performed by an independent team or company in addition to normal Verification and Validation (required for Cat B)
- OBSW Characteristics
- OBSW Process
  - OBSW Phases
  - OBSW Lifecycle
  - OBSW Verification
  - **OBSW Standards**
  - OBSW Documentation
- OBSW Architectures
- OBSW Environments
OBSW: Standards

European Cooperation for Space Standardization

Consultative Committee for Space Data Systems

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

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

On Board Software Overview for ULg

December 2017
- OBSW Characteristics
- OBSW Process
  - OBSW Phases
  - OBSW Lifecycle
  - OBSW Verification
  - OBSW Standards
- OBSW Documentation
- OBSW Architectures
- OBSW Environments
### OBSW : Documentation

#### Related file

- **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)
  - Software release document (SRelD)
  - Software user manual (SUM)
  - Software source code and media labels
  - Software product and media labels
- **DDF**
  - Software configuration file (SDD)
  - Software design document (SDD)
  - Software configuration file (SCF)
  - Software release document (SRelD)
  - Software user manual (SUM)
  - Software source code and media labels
  - Software product and media labels
- **MF**
  - Software configuration management plan (DRD in ECSS-M-ST-40)
  - Software review plan
- **PAF**
  - Software development plan
  - Software configuration management plan (DRD in ECSS-M-ST-40)
  - Software review plan
- **OP**
  - Maintenance plan
  - Migration plan
  - Software validation plan
  - Software verification plan
  - Software release document (SRelD)
  - Software user manual (SUM)
  - Software source code and media labels
  - Software product and media labels

#### DRL item

- (e.g. Plan, document, file, report, form, matrix)
- **DRL item having a DRD**
  - SRR
  - PDR
  - CDR
  - QR
  - AR
  - ORR

#### Diagram

- [Diagram showing software architecture and related documentation]
- OBSW Characteristics
- OBSW Process
- **OBSW Architectures**
- OBSW Environments
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
  - OBSW Functional Architecture
  - OBSW Static Architecture
  - OBSW Interfaces & Data Flows
  - OBSW Dynamic Architecture
  - OBSW Deployment Architecture
- OBSW Environments
OBSW: Functional Architecture (1/3)

- Functional Architecture

*definition of*
- *functional break down in*
  - functional modules*
  - functional interfaces* between these modules
On Board Software > Architecture > Functional Architecture > Functional Breakdown

**Functional Architecture (2/3)**

- **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

- **THERM** Thermal Management
  - for the temperature control of the spacecraft e.g. through thermal heater lines,

- **OBT** On-Board Time Management
  - For the synchronization of the on-board clock with ground time,

- **SM** Storage Management
  - for the storage of TM in case of Earth link unavailability

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

- **SMGT** Spacecraft Modes Management
  - handles the on-board system through the different mission phases defines the level of autonomy,

- **PWR** Electrical Power Supply Management
  - distributes the power coming from the battery and the solar arrays and manages battery charge / discharge,

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

- **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 spacecrafts e.g. through S-Band or X-Band links.

- **SADM** Solar Array Drive Mechanisms Management
  - for the optimal alignment of a satellite’s solar panels towards the Sun.

- **PL** Payload Management
  - for scientific payload or additional units
  - (very mission specific)

- **ETC … And Many Others**
  - And many other functions depending on the platform bus and on the mission

Note: interactions between functions are not depicted

Copyright © 2017 by SPACEBEL – All rights reserved
Functional Architecture (3/3)

- **Mission**
  - Mode Mgt
  - Processing
  - Commandability
  - Observability
  - Fault Mgt
  - SYS SW
  - PL SW
  - PF SW
  - AOCS SW

- **Operation**
  - Equipment Mgt
  - Processing
  - Telecommand Mgt
  - Telemetry Mgt
  - Fault Mgt

- **Avionics**
  - Network
    - Processor
  - Monitoring & Control Interfaces

- **Components**
  - Interaction Layer
  - Execution Platform

**On Board Software Overview for ULg**

December 2017
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
  ● OBSW Functional Architecture
  ● **OBSW Static Architecture**
  ● OBSW Interfaces & Data Flows
  ● OBSW Dynamic Architecture
  ● OBSW Deployment Architecture
● OBSW Environments
Static Architecture : Principles (1/4)

- Static architecture deals with
  - Functional Properties
  - Static Decomposition
  - Software Component Model

- Software is broken down
  - in software components (aka modules, units or objects)
  - with clear interfaces between them

- Components are organized (see slide 2/4)
  - horizontally in variability layers with coherent abstraction level and
  - vertically in functional chains with coherent functionality

- Component are functional units (see slides 2-3-4/4)
  - That encapsulate functional services,
  - that expose these services to other components through well defined interfaces, and
  - that can be assembled together to build a software product

- Components are made up of (see slide 3/4)
  - Algorithms, State Machines
  - Data Structures

- Major design decision is (see slide 4/4)
  - Central Data Pool vs
  - Distributed Data Flows

- Main programming paradigms are
  - Procedural programming vs
  - Object Oriented programming (not used so far in on board software)
On Board Software Overview for ULg

- **Mission Dependent**
- **Operation Dependent**
- **Avionics Dependent**
- **Platform Dependent**

**FUNCTIONAL CHAINS**
- Thermal Mgr
- Power Mgr
- FDIR
- AOCS
- SC Mgr
- ...

**INTERFACES**

**VARIABILITY LAYERS**

Coherent Abstraction Level

Coherent Functionality

**December 2017**
Static Architecture: Components (3/4)

On Board Software > Architecture > Static Architecture > Component

COMPONENT
MODULE
UNIT
OBJECT

Provided Interface

Data Structures

State Machines

Algorithms

Functions
Procedures
Methods

Required Interface
Static Architecture (4/4)

Central Data Pool VS Distributed Data Flow
Static Architecture: Overview

Design Pattern (« a repeatable solution ... »)

Middleware (« a connectivity service ... »)

Common Services (« shared services ... »)

Lightweight On Board Application Framework

Applications

Mission Specific Applications

Generic Component

Platform Specific Components

Layered and Modular Architecture

PUS services

On Board File System

Core Services

On Board Procedures Interpreter

Equipment & Resource Managers

DHSW Support Package

RTOS I/F

Device Drivers

BSP

RTOS

HDSW

December 2017

On Board Software Overview for ULg
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
  - OBSW Functional Architecture
  - OBSW Static Architecture
  - OBSW Interfaces & Data Flows
- OBSW Dynamic Architecture
- OBSW Deployment Architecture
- OBSW Environments
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

PF
Equipment

Command

Control

OBSW
On Board Software

Space Ground Interface

Command

Control

PL
Payloads

Science Data

GND
Control Center

Mission Equipment

Platform Interfaces

Telecommand

Telemetry

Payloads Interfaces

December 2017

On Board Software Overview for ULg
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
  ● OBSW Functional Architecture
  ● OBSW Static Architecture
  ● OBSW Interfaces & Data Flows
  ● **OBSW Dynamic Architecture**
  ● OBSW Deployment Architecture
● OBSW Environments
Dynamic Architecture : Principles  (1/3)

- Dynamic Architecture deals with
  - Non Functional Properties
  - Dynamic Behaviour
  - Computational Model

- Main concepts are (see slide 2/3)
  - Tasks, Threads, Traps, Interrupts
  - Scheduling, Processing Time, Execution Time, Deadlines, Priorities
  - Communication, Synchronisation (Critical Sections, Mutual Exclusion)

- Major design decision is (see slide 3/3)
  - Periodic/Cyclic/Synchronous/Time Driven/Polling vs (Period, Offset)
  - Aperiodic/Sporadic/Asynchronous/Event Driven/Interrupts (Minimum Inter Arrival Time)

- Typical problems are
  - Race Conditions, Deadlock Starvations, Livelocks
Dynamic Architecture: Concepts

Scheduling
- Periodicity
- Priorities, Preemption

Tasking
- Periodic Tasks

Shared Executable
- (reentrance of code ...)

Shared Resources
- (memory, inputs/outputs, ...)

Synchronisation
- Semaphores
- Mutexes
- Critical Sections

Communications
- Pipes, sockets
- Message queues, mailboxes, ...

Interrupts
- Hardware Interrupts
- Software Traps

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

On Board Software Overview for ULg
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
Dynamic Architecture: Schedulability

- Proof of software schedulability
  - 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 or measured)
  - Processor utilization
  - Worst Case Execution Times (WCET)

- Memory Footprint
- Stack Usage
Dynamic Architecture: Schedulability

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.
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
  - OBSW Functional Architecture
  - OBSW Static Architecture
  - OBSW Interfaces & Data Flows
  - OBSW Dynamic Architecture
  - **OBSW Deployment Architecture**
- OBSW Environments
Deployment Architecture: Principles

- Deployment Architecture deals with the Mapping
  - of a logical architecture (software components)
  - to a physical environment (hardware resources).

- Main concepts are
  - Distribution
  - Communication

- Main design decision is
  - Centralised Architecture vs
  - Distributed Architecture

- Main Solutions are
  - Multi Processor
  - Multi Core
  - Time and Space Partitioning
Deployment Architecture: 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.
Deployment Architecture: SCM

Software Component Model

Platform Independent Model

Design Level

Functional Concerns (algorithms)

Required Interface

Component

Connector

Container

Execution Platform

Computational Model

Composition

Reusability

Replaceability

Platform Specific Model

Implementation Level

Non Functional Concerns (tasking, timing, config, init, security, …)

Provided Interface

Component

Container

Separation of Concern

Pairing up

December 2017

On Board Software Overview for ULg
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
- **OBSW Environments**
  - OBSW Execution Environment
  - OBSW Development Environment
OBSW Environment

Cross Development Environment

Development Environment:
- Powerfull Development Workstations
- Confortable Operating System
- Production Tools

Execution Environment:
- On Board Computer with Limited Resources
- Real Time Operating System with Constraints
- Software Executable
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
- OBSW Environments
  - OBSW Execution Environment
    - Processors
    - Operating Systems
  - OBSW Development Environment
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
- OBSW Environments
  - OBSW Execution Environment
    - OB Processors
    - OB Operating Systems
  - OBSW Development Environment
Exec Environment: Processors

- Hardened Space Qualified processors
  1990: 1750 – 16 Bits – 2 MIPS
  2010 : LEON2 – SPARC V8 - 32 Bits – Cache – Pipeline -100 Mhz – 84 MIPS
  2105 : LEON3/4– Quad Core LEON

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

See Avionics Overview
● OBSW Characteristics

● OBSW Process

● OBSW Architectures

● OBSW Environments
  ● OBSW Execution Environment
  ● OB Processors
  ● **OB Operating Systems**

● OBSW Development Environment
Exec Environment: Operating System

- **RTOS**: Real Time Operating Systems
  - VxWorks (Commercial)
  - RTEMS (Open Source)
  - Linux RT (Not widely used … in OBSW)

- **TSP**: Time and Space Partitionning
  - Hypervisor
  - µKernel
- OBSW Characteristics
- OBSW Process
- OBSW Architectures
- OBSW Environments
  - OBSW Execution Environment
  - **OBSW Development Environment**
    - Modelling
    - Programming
    - Production
    - Validation
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
● OBSW Environments
  ● OBSW Execution Environment
  ● OBSW Development Environment
  ● OBSW Modelling
● OBSW Programming
● OBSW Production
● OBSW Validation
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.
Development Environment: Modeling

- **Paradigm Shift**
  - From Programming to Modelling
  - Model Based Software Engineering

- **Modeling Domains**
  - Requirements
  - Architecture Modeling,
  - Data Modeling,
  - Behaviour Modeling

- **Modeling Languages**
  - UML, SYSML, AADL, AAML, SDL, …

- **Model Verification**
  - Strong Syntax, Formal Verification

- **Automatic Generation**
  - Code
  - Documents

---

**ECLIPSE**
Integrated Development Environment (IDE) and Eclipse Modelling Framework (EMF)
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
● OBSW Environments
  ● OBSW Execution Environment
  ● OBSW Development Environment
  ● OBSW Modelling
  ● **OBSW Programming**
● OBSW Production
● OBSW Validation
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
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
● OBSW Environments
  ● OBSW Execution Environment
  ● OBSW Development Environment
  ● OBSW Modelling
  ● OBSW Programming
● **OBSW Production**
● OBSW Validation
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
● OBSW Characteristics
● OBSW Process
● OBSW Architectures
● OBSW Environments
  ● OBSW Execution Environment
  ● OBSW Development Environment
    ● OBSW Modelling
    ● OBSW Programming
    ● OBSW Production
● OBSW Validation
Development Environment: Validation

On Board Software > Environment > Validation Environment

December 2017
OBSW

Modelling
OBSW Modeling

- Modeling Examples
- Modeling Definition
- Modeling Objectives
- Modeling Characteristics
- Modeling Methods
Modeling : Examples

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.
What is a Model?

- A model is a **description** of (part of) a system written in a **well defined language**. A well defined language is a language with a well defined form (syntax), and meaning (semantic), which is suitable for an automatic interpretation by a computer. (Anneke Kleppe et al. « MDA Explained »)

- A formal **representation** of a function, behaviour and structure of the system we are considering. Expressed in an unambiguous language. (Chris. Raistrick et. Al. « Model Driven Architecture and Executable UML »)

- Models are an **abstraction** of the reality captured in a specific representation format i.e. diagram or language.

- A model is a **simplification** of something so we can view, manipulate, and reason about it, and so help us understand the complexity inherent in the subject under study. (Steve Mellor et. All « UML Distilled »)

- A simplification of a system built with an intended goal in mind: the model should be able to answer questions in place of the actual system. (J. Bézivin & O. Gerbé. « Towards a precise définition of the OMG/MDA Framework »)

- A model is a complex structure that represents a design artifact such as a relational schema, and interface definition (API), and XML schema, a semantic network, a UML model or an hypermedia document (Phil. Bernstein. « A Vision of Management of Complex Systems »)

- A model captures a **view** of a physical system. It is an abstraction of a physical system with a certain purpose; This purpose determines what is included in the model and what is relevant. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail. (OMG. « UML Superstructure »)

- A functional specification of the function, structure and/or behaviour of an application or system. (OMG. « MDA Guide »)
Objectives of Software Modeling

- To deal with complexity of systems development through
  - Abstraction: Abstract a problem to focus on some particular points of interest and to improve understandability of a problem
  - Iteration: Iterative modeling may be expressed at different level of fidelity
  - Separation of Concerns: Possible set of nearly independent views of a model (“Aspect Oriented Modeling”)
  - Domain Specific Language: To focus on specific domain expertise

- To minimize development risks through
  - Through analysis and experimentation performed earlier in the design cycle
  - Enable to investigate and compare alternative solutions

- To improve communication ...
  - to foster information sharing and reuse!
  A model is often best suited than a long speech!
Characteristics of Useful Models:

- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Inexpensive**
  - Much cheaper to construct and study than the modeled system

To be useful, engineering models must satisfy all of these characteristics!
In an attempt to formalize more and more the expression of the software documentation and production, the design, architecture and requirements have moved from simple text to drawings and from drawings to models.

With the emergence of new tools these model representations can be constructed, translated and exploited in different ways:

- **Analysis**: various types of model checking e.g. completeness and consistency analysis
- **Simulation**: execution of model, behavior can be simulated
- **Design**: decomposition of system in smaller components, establishing interfaces
- **Coding**: automatic generation of source code e.g. C or Ada (auto-coding)
- **Testing**: automatic generation of tests (auto-testing)
- **Proving**: formal verification or proof
Modeling: Methods

- Modeling Methods
  - **UML** (Unified Modelling Language)
    Not a method (weak semantic) but a notation (well defined syntax)
    Can be extended by profiles (e.g. SysML, MARTE, …)
    Can be complemented/supplemented (e.g. OCL)
  - **AADL** (Architecture Analysis and Design Language)
  - **SDL** (Specification and Description Language)
  - …. and many others (e.g. AAML …)
  - See also Matlab/Simulink/Stateflow

December 2017
Use Case Diagram

On Board Software Overview for ULg

CFDP manager

PUS services
configure, control and monitor CFDP transactions
perform high-level operations on the distributed CFDP file-system
transfer PUS messages to a remote SC

Communication system
send and receive PDUs to and from remote CFDP entities

remote CFDP entity
remote file system
local CFDP entity
local file system
inter spacecraft link
space link

onboard application
local CFDP entity

December 2017
On Board Software Overview for ULg
Collaboration Diagram

On Board Software > Modeling > UML > Collaboration Diagram
Class Diagram

On Board Software > Modeling > UML > Class Diagram

December 2017

On Board Software Overview for ULg
In this example, there is still enough memory resources for cfdp_transaction and cfdp_receiver but not for the bit array.
State Diagram

On Board Software > Modeling > UML > State Diagram

- link available
  - stopped
    - stop
    - start
  - started
    - running
      - suspend
        - resume
        - delay elapsed
    - suspended
    - fired
  - not locked
    - buffer ready
    - set to available
      - not locked
        - start transfer
        - set to available
        - set to unavailable
      - locked
        - start transfer
  - locked
    - buffer ready
    - set to available
      - locked
        - start transfer
        - set to available
        - set to unavailable
      - non available
        - not locked
          - buffer ready

December 2017

On Board Software Overview for ULg
OBSW

On Board Control Procedures
What are OBCPs?

- On Board Control Procedures (OBCPs) are flight procedures written in a high level language.

- These are pieces of software that can be resident on board or that can be uploaded to the spacecraft as required by ground,

- to be interpreted and executed on board, on demand, at any time and

- that may interact, to different extents, with the rest of the data handling system.

- They differ from native applicative components in that their invocation and execution may be controlled. This concerns in particular the ability to suspend or abort their execution.
Why OBCPs?

- OBCPs provide a flexible way to operate the spacecraft, to extend the on board software functionality or to modify the behaviour of on board applications, even in flight.

- OBCPs participate to on board autonomy when a rapid reaction is needed in spite of reduced spacecraft visibility or long propagation delay.

- Trend is to have generic functions implemented in the FSW and mission specific functions in OBCPs.
# What are OBCP Used For?

<table>
<thead>
<tr>
<th>Type</th>
<th>Applications</th>
<th>Interaction</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBOP</td>
<td>- On Board Execution of Ground Procedures, - End-of-Life Operations</td>
<td>Limited to - Telecommand - Telemetry - Event</td>
<td>OP Engineer</td>
</tr>
<tr>
<td>On Board Operation Procedures</td>
<td>- Mode Management, - Configuration Sequences, - Recovery Procedures</td>
<td></td>
<td>PF or PL Engineer</td>
</tr>
<tr>
<td></td>
<td>- Support to Assembly Integration and Testing - Long and complex configuration sequences</td>
<td></td>
<td>AIT Engineer</td>
</tr>
<tr>
<td></td>
<td>- Fault injection and robustness testing - Temporary functions for testing purposes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBAP</td>
<td>- Simple On Board Applications - Mission Specific Functions - Accomodation of Late Definition</td>
<td>Extended to - Data Pool - Equipments - Drivers</td>
<td>PF or PL Engineer</td>
</tr>
<tr>
<td>On Board Application Procedures</td>
<td>- Debugging - Short Term Work Around Solutions - On Board Patches - Adaptation to unpredictable environment</td>
<td></td>
<td>OBSW Engineer</td>
</tr>
</tbody>
</table>
How do OBCPs work?

OBCPs are executed on board.

The execution on board relies on virtual machine that interprets the instructions and interacts with the rest of the On Board Software while also providing some kind of time and in space isolation for fault containment.

OBCPs are uploaded on board by ground

The intermediate byte code is uploaded via appropriate telecommands and is stored on board.

OBCPs are first prepared on ground

The preparation on ground relies on an environment composed amongst others of an editor, a compiler and, ideally, a debugger.

The OBCPs are written in a high level user language. They are first compiled and linked on ground to yield an intermediate byte code that can efficiently be uploaded and interpreted on board.
OBCP Overall Architecture

- OBCP User Language
- OBCP Ground Preparation Environment
- OBCP Intermediate Byte Code
- OBCP On Board Execution Environment

- OBCP Ground Debugger Environment

December 2017
On Board Software Overview for ULg
OBCP Ground Dev Env Architecture

OBCP User Language

OBCP Ground Development Environment

OBCP Target Code

OBCP On Board Execution Environment

OBCP Ground Debugger Environment

OBCP Editor

OBCP User Language

SDB interface

OBCP Compiler

OBCP Converter

OBCP Target Code

Satellite Data Base

Cat D SW
OBCP OB Exec Env Architecture

- OBCP User Language
- OBCP Ground Development Environment
- OBCP Target Code
- OBCP Ground Debugger Environment
- OBCP On Board Execution Environment
- OBCP Target Code
- OBCP Manager
- OBCP Interpreter
  - OBCP Scheduler
  - OBSW Interface
- On Board Software

Cat C/B SW
OBCP Overall Architecture

- OBCP Editor
- OBCP User Language
- SDB interface
- OBCP Compiler
- OBCP Converter
- Satellite Data Base
- OBCP Target Code
- OBCP Manager
- OBCP Interpreter
- OBCP Scheduler
- OBSW Interface
- On Board Software
- Onboard Execution Environment

Ground Development Environment
- OBCP Interpreter
- OBSW Interface
- OBSW Stubs

Ground Debugging Environment
- OBCP Debugger

On Board Software Overview for ULg