Model-driven & component-based engineering

Developping the OBSW of an autonomous satellite

Jérémie POULY (CNES)

ADCSS 2014

SOMMAIRE

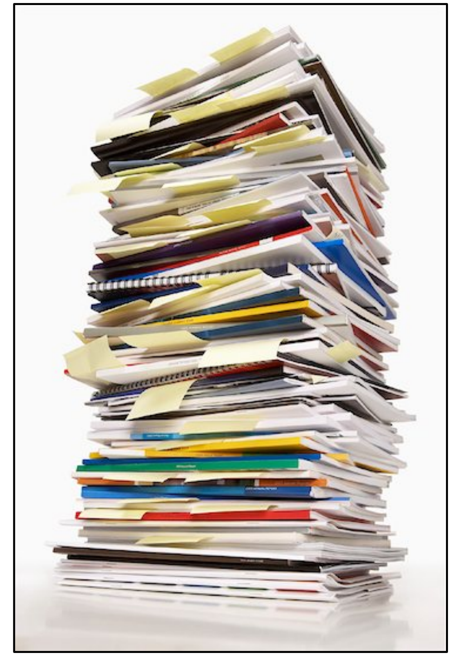
CONTEXT

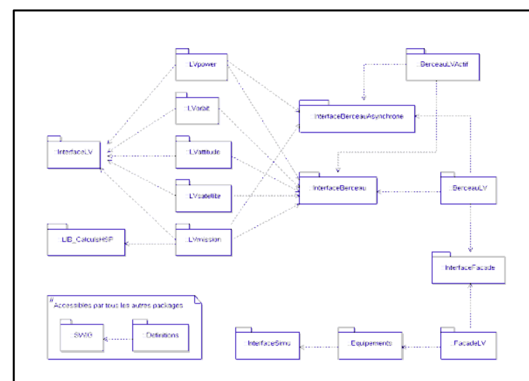
- System issues
  - Increase level of on-board autonomy
  - Increase hardware independency
  - Increase software flexibility
- Increasing on-board processing power

time

?

?



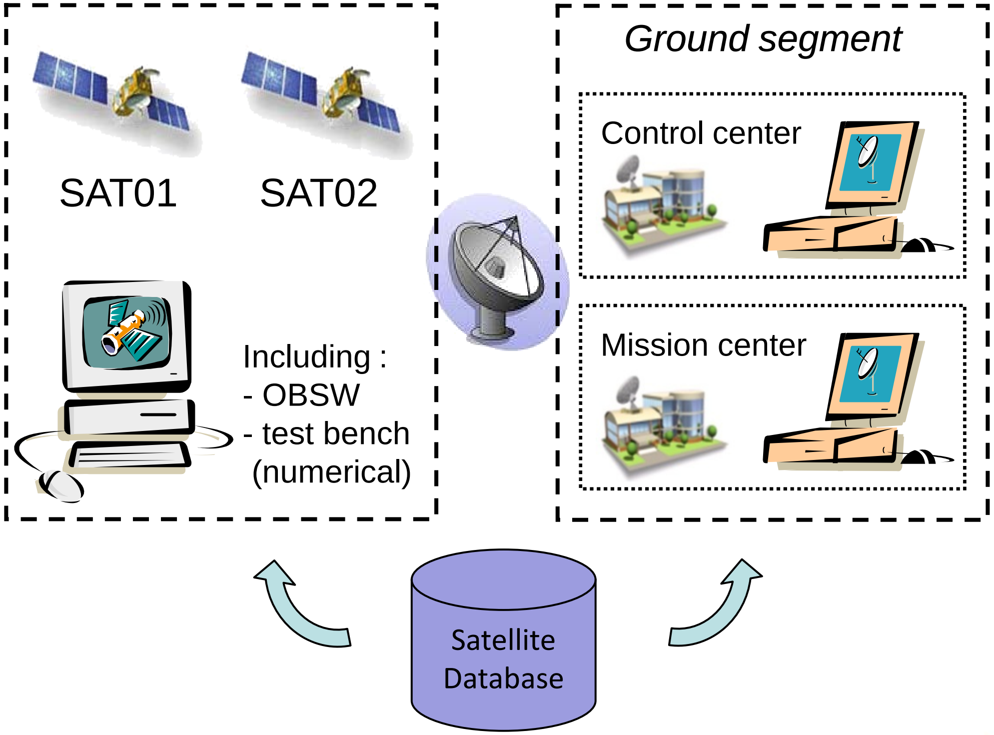


➢ New methods are required for On-Board SoftWare (OBSW) development

SOMMAIRE

AGATA PROGRAM

- AGATA « Autonomy Generic Architecture – Tests and Applications »
  - CNES-ONERA joint research program decided in 2004
  - **Develop a ground tool to demonstrate the feasibility and interest of autonomy for space systems**
  - Define and test a process for the development and the validation of OBSW dedicated to autonomy
  - Develop a rapid prototyping tool to evaluate autonomy concepts for future projects
  - Multi-domain activity
    - Several CNES departments involved
    - Onera, Thales Alenia Space, Airbus DS, LAAS, Alten, CSSI, Spacebel…



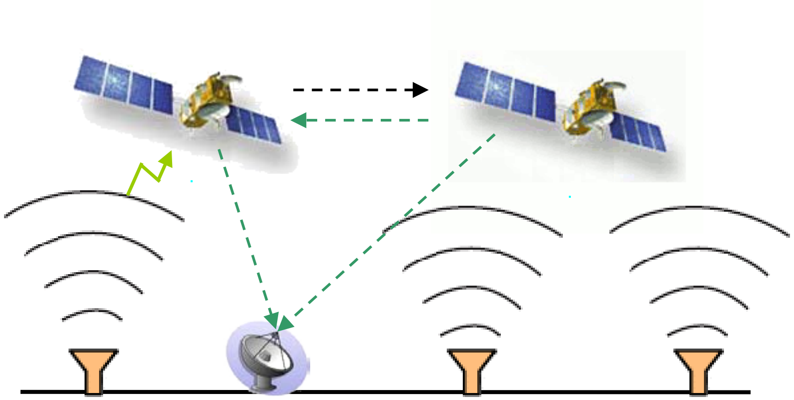
AGATA GROUND DEMONSTRATOR

- A demonstrator for on-board autonomy
  - Fully autonomous missions
- A demonstrator for new technologies
  - UML specification of the OBSW (Eclipse/Topcased)
  - MDE (automatic code generation)
  - RTSJ (“Real-Time Specification for Java”)
  - PUS, OBCP
- Based on 3 independent components
  - A generic satellite simulator (BASILES)
    - Fully numerical or hybrid simulation (TSIM-HW “Leon2” CPU board)
  - A ground segment (OCTAVE)
    - PUS-compliant (CNES tailorisation of ECSS E70-41A)
  - A specific OBSW, based on a generic component-oriented architecture
    - PUS-compliant (CNES tailorisation of ECSS E70-41A)
    - On-board mission planning algorithm

AGATA-ONE MISSION

- Reference scenario « AGATA-ONE »
  - Earth monitoring mission (two LEO Agile satellites)
  - Autonomously detect, record and download the data of sources of electromagnetic emission (SEE) around the Earth
  - Record and download ground requests
  - Highly variable and unpredictable data rate
    - between sources
    - for a single source with respect to time
  - Large amount of data collected
  - But limited downlink capacity

  - Requires on-board reactive planning to maximize data download



SOMMAIRE

TOWARD A NEW PARADIGM

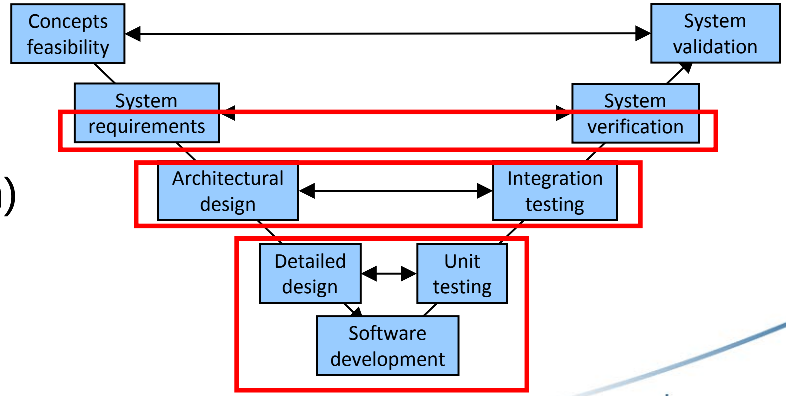
- Autonomous OBSW complexity
  - Impossible to foresee all feasibility dead ends
  - Necessity to prototype complex functions and interactions
- Validation issues (usual validation methods inappropriated)
  - Decision tree far too complex
  - System reaction not necessarily predictable
  - ➢ Exhaustive testing is impossible
- Hardware abstraction

➢ How to design the autonomous AGATA OBSW ?

- Define a generic modular architecture
  - Building-blocks approach: “divide and conquer” paradigm
- Follow an iterative and incremental development process
  - Validation process nested in development process

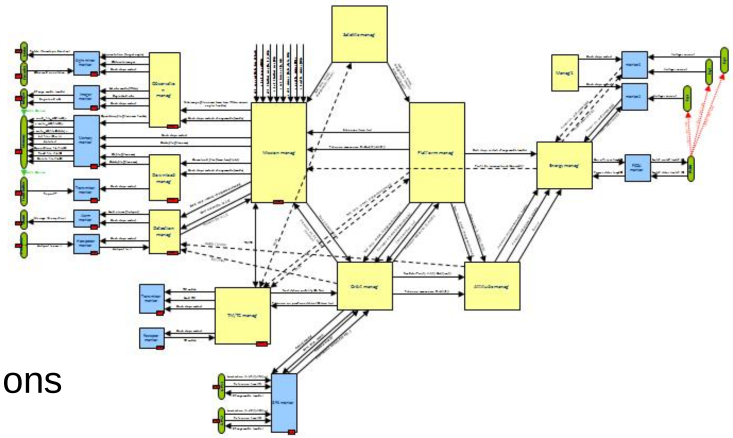
MODEL DRIVEN ENGINEERING

- Modeling environment
  - Language ➔ UML 2.0
  - Modeler ➔ Eclipse/Topcased
- UML specification can be validated early in the development process
- Software UML specification is the heart of the process
  - Software architecture (class, interface and package)
  - Dynamic behavior (composite structure, state machines and activity)
- Customized UML profile for non-functional properties (AutoJava)
- Associated to Java/RTSJ code generator
  - Real-time RTSJ code execution on RTEMS/Leon (AeroVM)
  - Functional Java code execution
- Model-based development process enhancements (“Y-shaped” life cycle)
  - Validation (tests generation, model simulation)
  - Implementation (code generation)

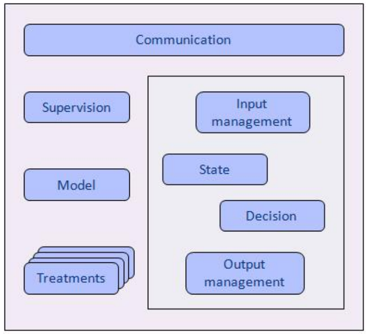


COMPONENT-BASED ENGINEERING

- Generic modular architecture adapted to advanced autonomy needs
  - Theoretical preliminary architecture designed by Onéra
    - All modules built on same pattern
    - Each module in charge of a function
    - Hierarchical module organization
    - Low-coupling between module
  - Decision process combines two tasks
    - Reactive control task (compatible with real-time constraints)
    - Deliberative reasoning task for autonomy functions



- “Building blocks” approach
  - Early interface definition between main functions
  - Independent validation of OBSW components
  - Architecture inspired from COrDeT recommandations
- COTS (Components Of The Shelf) approach
  - Re-use of previously developped/validated components

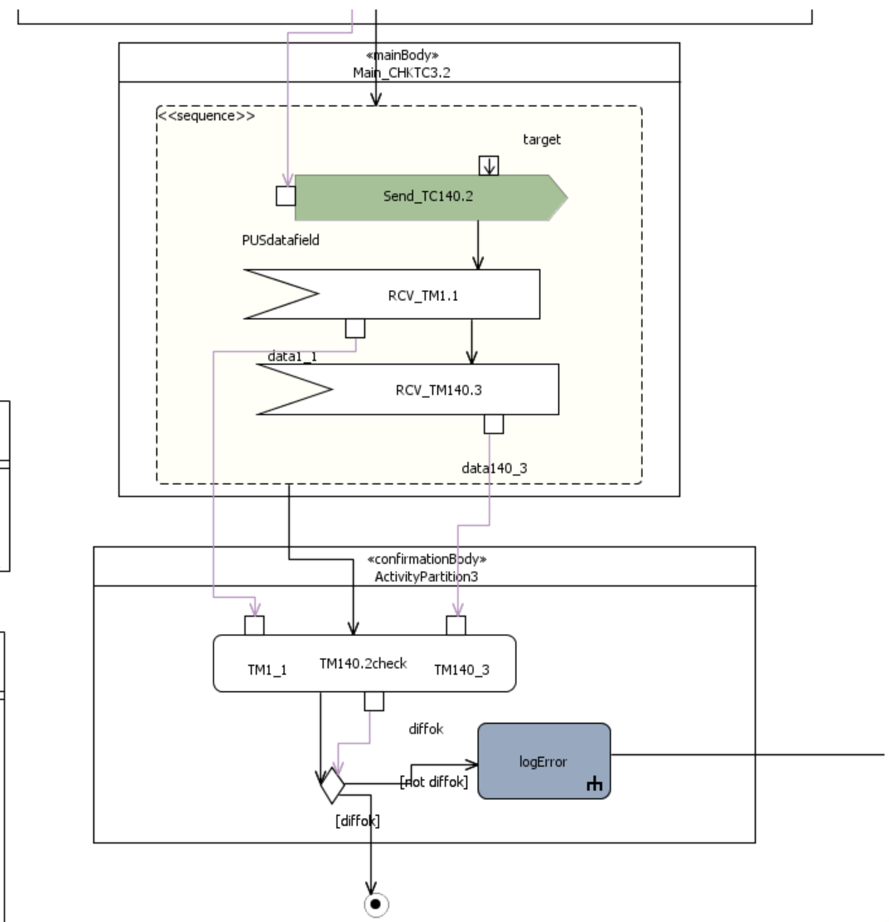


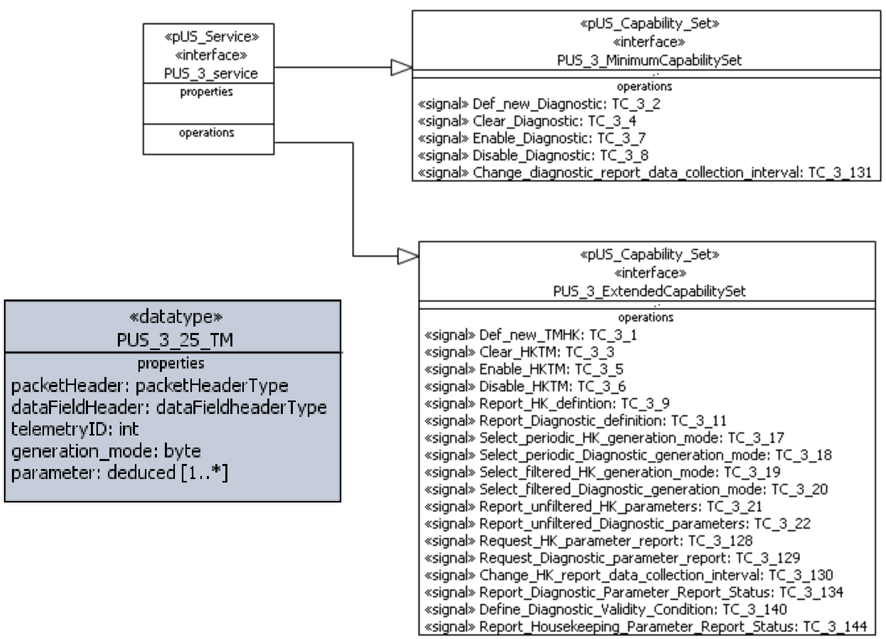
SOMMAIRE

- INTRODUCTION
- AGATA PROGRAM
- MDE & CBSE
- DEVELOPMENT PROCESS
- OBSW ARCHITECTURE
- CONCLUSION

VALIDATION TESTS GENERATION

- “Black-box” testing (TM/TC external interface)
- Dedicated TM/TC model independent from design model
- Bench I/F for real-time tests execution with final OBSW
- Two generation levels
  - Unit tests (OCL language)
  - Tests sequences (activity diagrams)

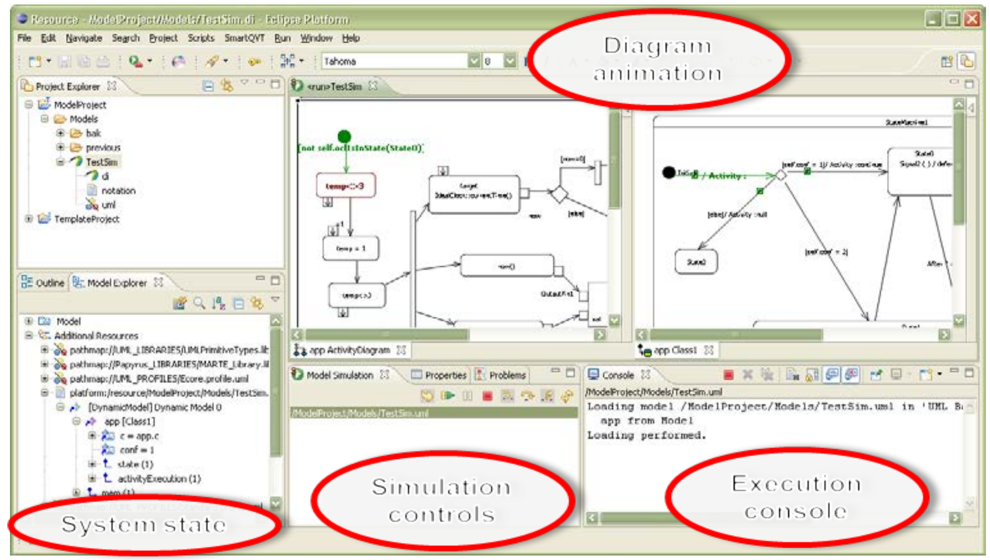


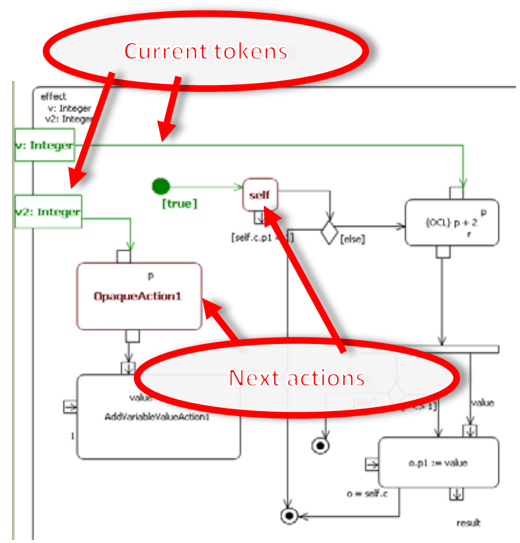


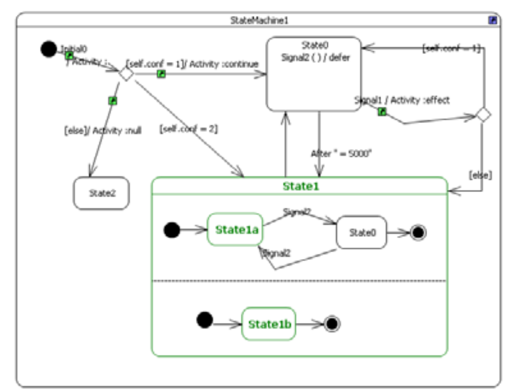
MODEL SIMULATION

- Simulates model behavior based on state machines and activity diagrams
- “Topcased-Simu” module
- Model-debugging & more
  - Early model simulation
  - “Co-simulation”
- Standard debug interface

- External events (TC,…)
  - W.r.t. selected object
  - Smart filtering I/F
- Creates a dynamic model
  - From configuration files
  - From composite structure diagrams
- Diagrams of dynamic model are “animated” in real-time

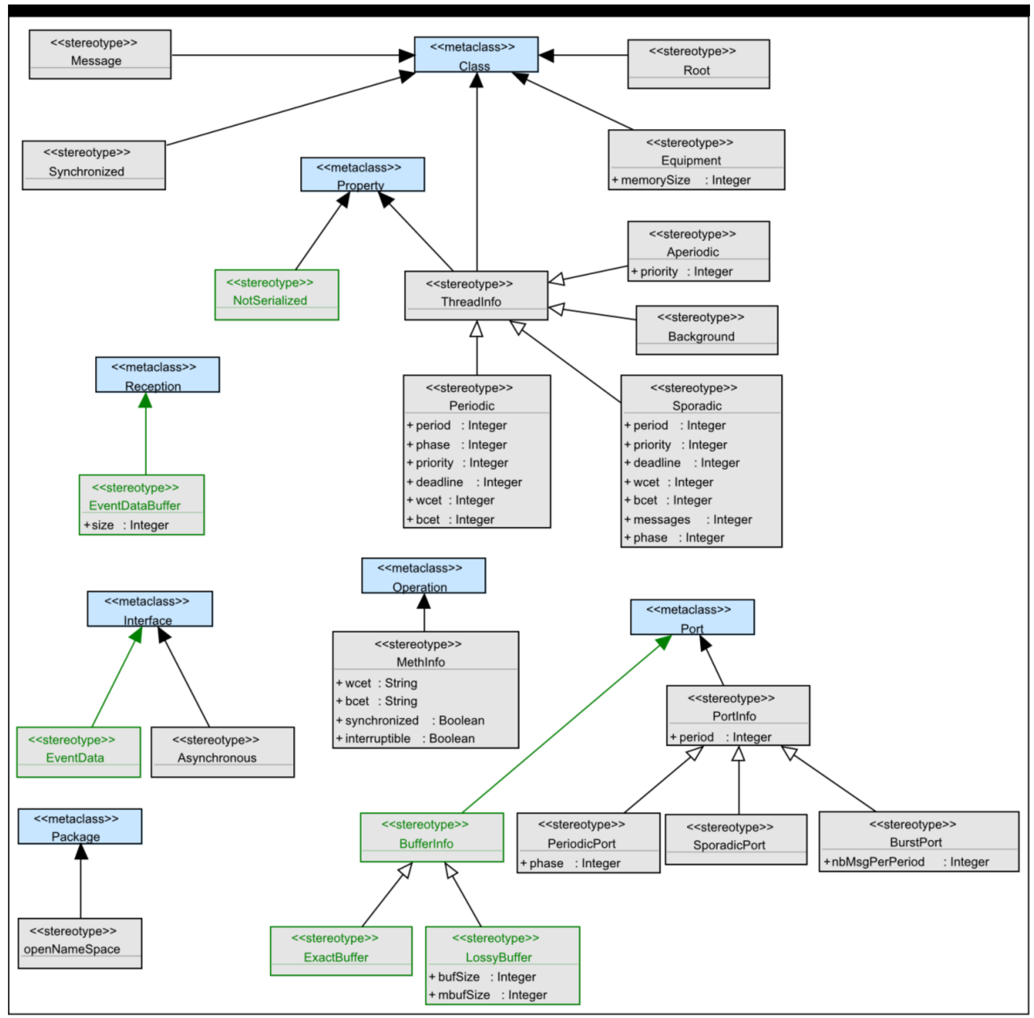






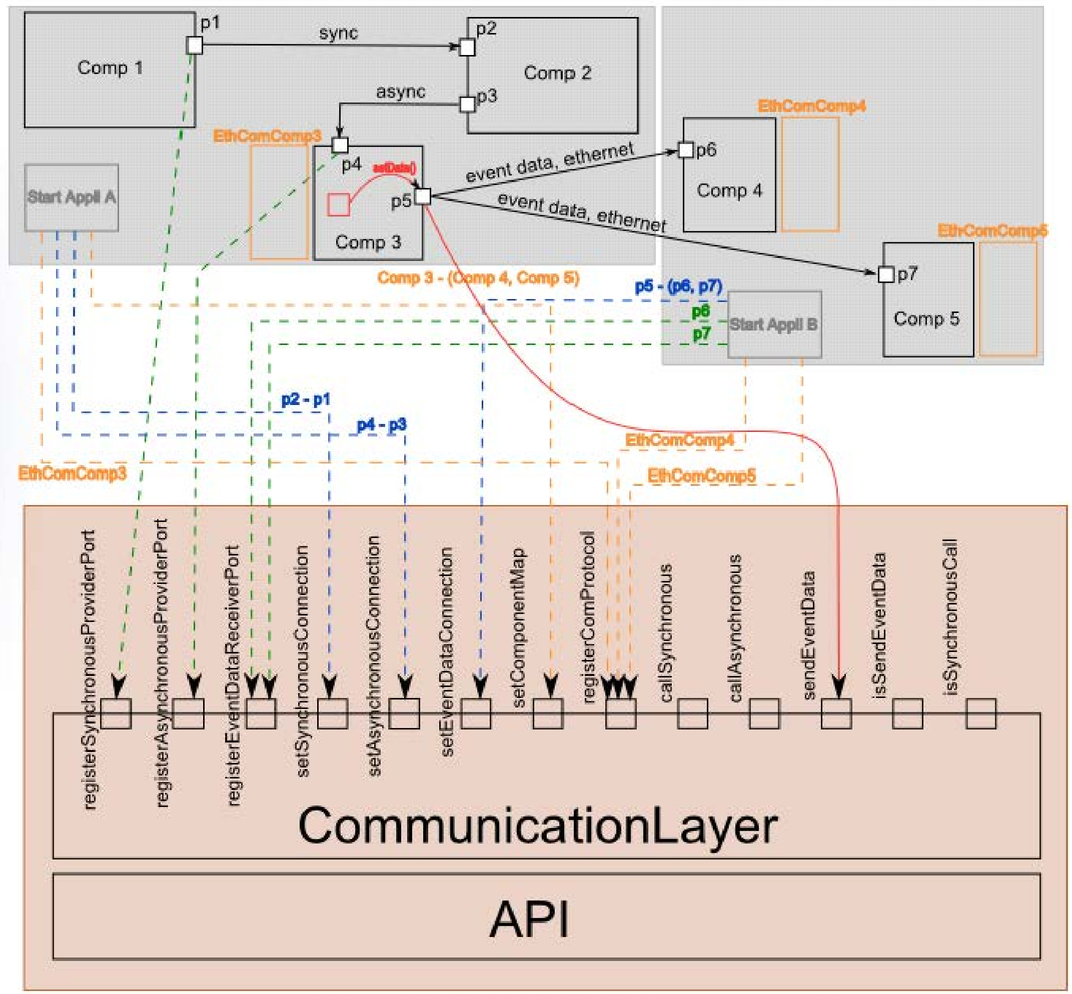
AUTOMATED CODE GENERATION (1/5)

- Java / RT-Java (RTSJ) generation capability
- Historic CNES UML profile "autojava" for NFP
  - Tasks real-time properties
  - Ports and buffers properties
  - Shared data protection
  - WCET…
- "uml2rtsj" code generator
  - "UML generators" project in PolarSys Eclipse IWG
  - Based on Acceleo 3
  - Features supported
    - Class/interface/package
    - State machines
    - Composite structure



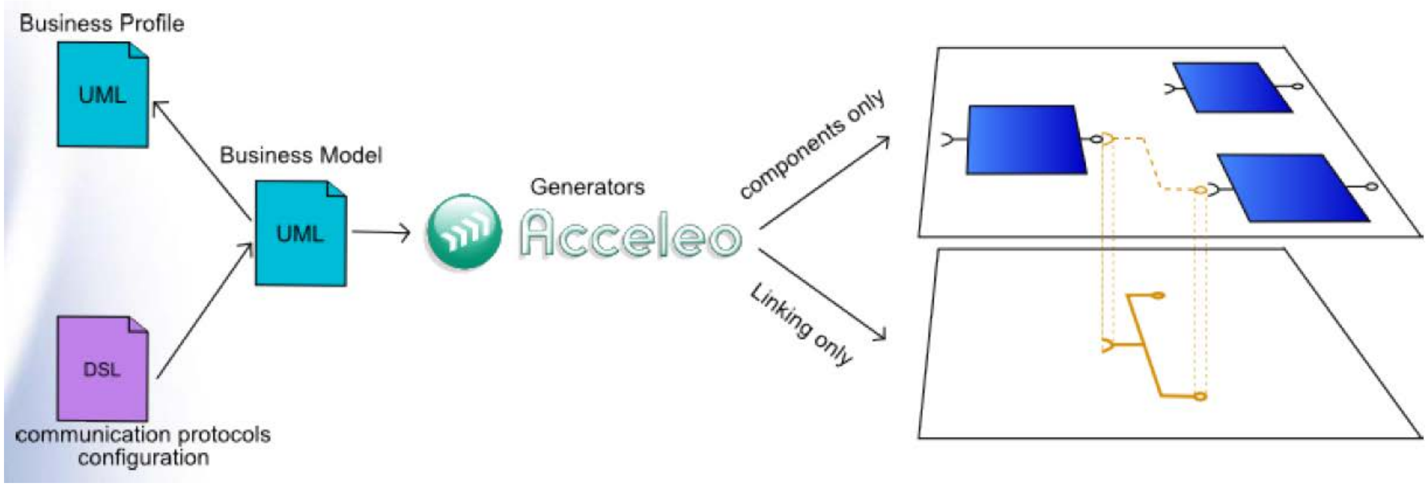
AUTOMATED CODE GENERATION (2/5)

- Generates a component oriented architecture
- Inspired from COrDeT
  - SOIS Message Transfer Service implemented
  - Interaction layer entirely generated from model
  - 2 « execution plateforms » supported
    - Classic JRE
    - AeroVM + RTEMS
- Limitation wrt COrDeT
  - Reduced set of NFP
  - Components design and components interactions in the same model
  - …



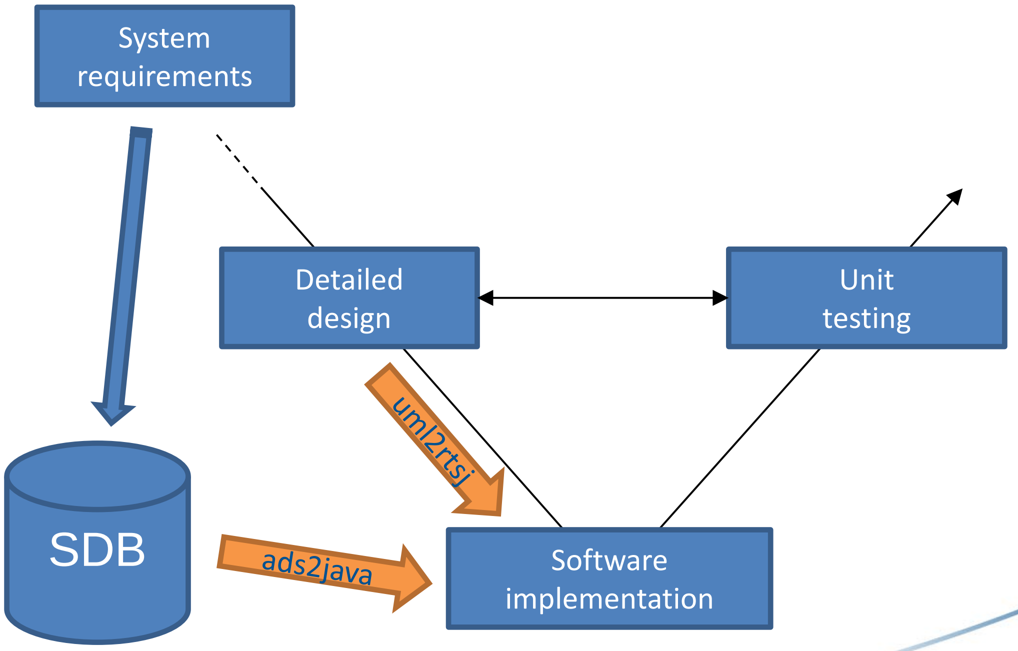
AUTOMATED CODE GENERATION (3/5)

- Standalone component generation
- Independent middleware generation
  - Direct or Ethernet communication supported
  - Extensible to other communication protocols
  - Configurated via model decoration
    - Separate functional from technical preoccupations
    - Benefit from several configurations on the same model



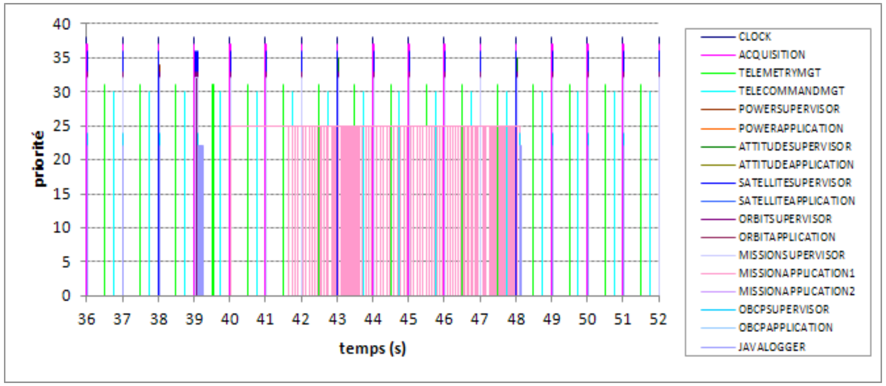
AUTOMATED CODE GENERATION (4/5)

- Satellite DataBase synchronisation
- SDB often evolves after OBSW developments have started
- SDB contains lots of information used by the OBSW
  - TM/TC packets definition and tailoring
  - Value of many configuration parameters
  - PUS specific information (parameter ID of all SW variables,…)
- All relevant information in SDB is extracted by automated code generation
  - Specific SDB to Java generator
  - SDB changes are automatically integrated in the OBSW



AUTOMATED CODE GENERATION (5/5)

- Code generator easy to evolve thanks to Acceleo
  - Several interaction layer improvements to access to system demands
    - Asynchronous signals priorities
    - Communication errors handling
    - Sub-states management
- "RT-Java" code associated execution plateform
  - AeroVM (adaptation of JamaicaVM made by AirbusDS/Aicas for ESA)
  - RTEMS + Leon TSIM-HW board
  - Real-time evaluation of the autonomous AGATA OBSW (hybrid simulation)
    - CPU needs compatible with today's standards (Leon2)

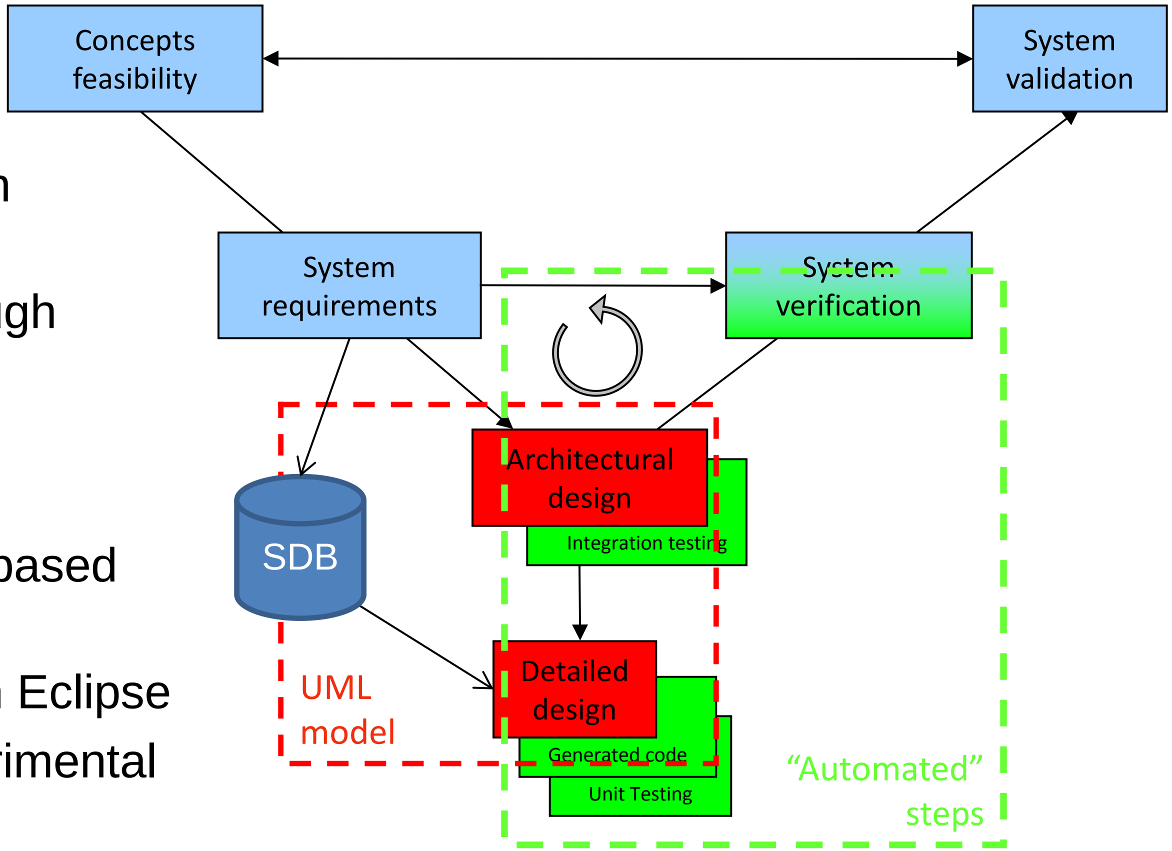


- Integration of OPISS virtual machine (OBCP engine)
  - 2 components dedicated to OPISS developed independently
  - Merging « rendez-vous » organized on a monthly basis
    - Mainly to integrate SDB-related evolutions

RESULTING "Y" LIFE CYCLE

- Based on UML software specification
- Reduced software production time through
  - Tests generation
  - Model simulation
  - Code generation
- Support component-based engineering
- Process integrated in Eclipse

… but remains experimental

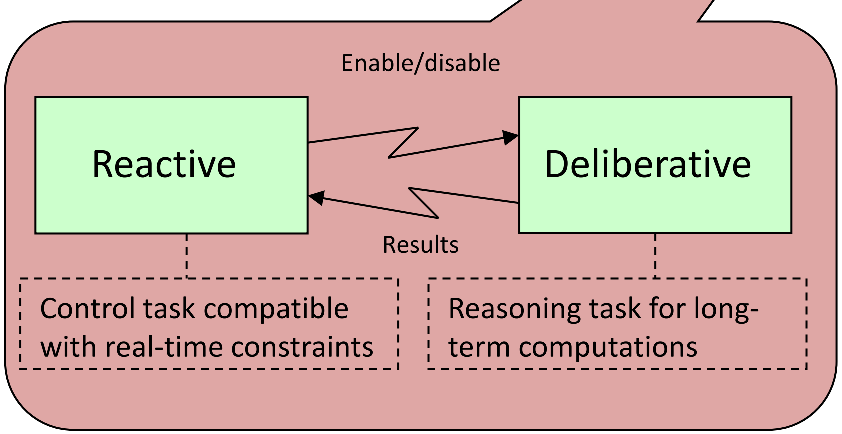


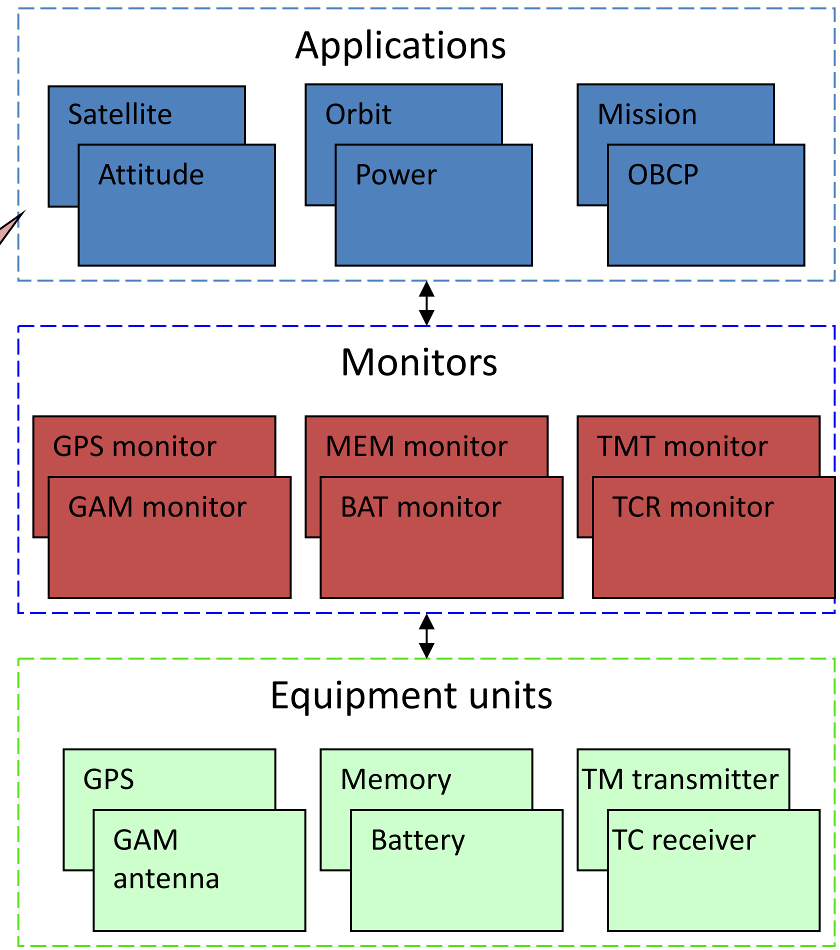
Model driven engineering relies extensively on tooling

SOMMAIRE

AGATA OBSW ARCHITECTURE

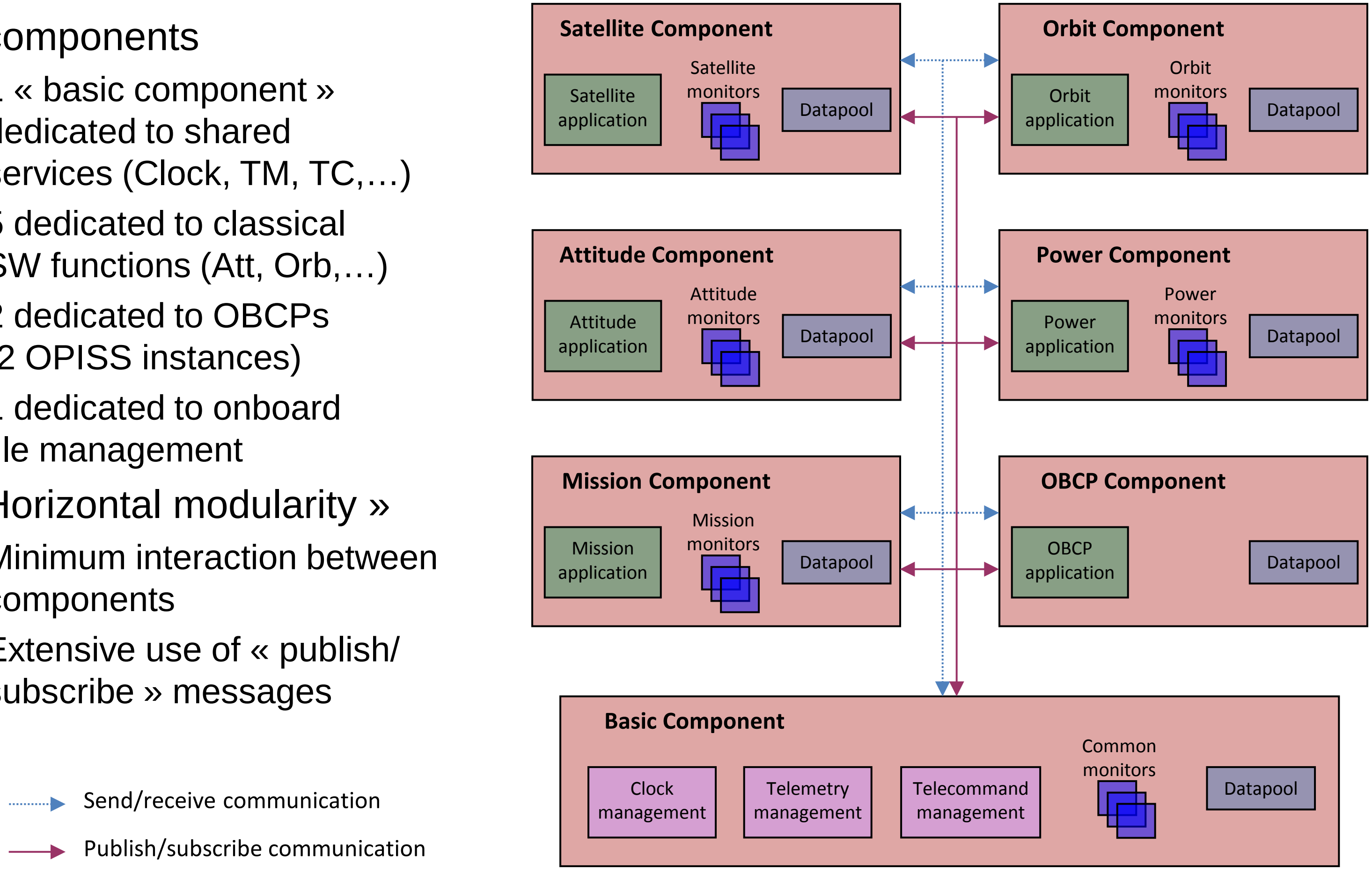
- Hierarchical functional architecture
- « Monitor » layer
  - Abstraction of equipment units
  - « Vertical modularity »
- Application processes
  - Reactive/deliberative decision process





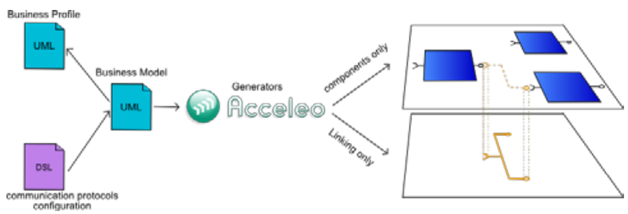
« AGATA » COMPONENT-BASED APPROACH

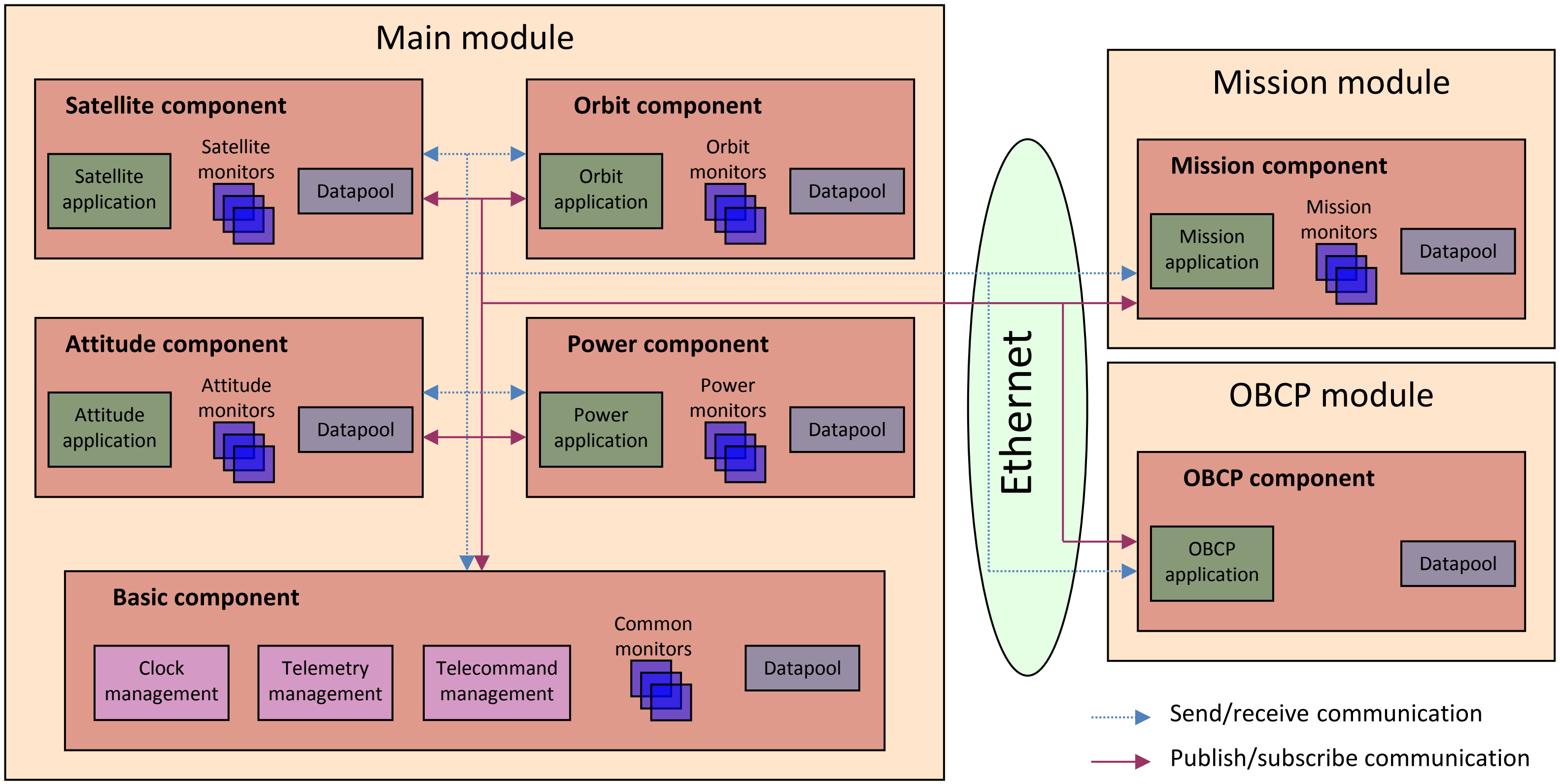
- OBSW architecture
  - 9 components
    - 1 « basic component » dedicated to shared services (Clock, TM, TC,…)
    - 5 dedicated to classical SW functions (Att, Orb,…)
    - 2 dedicated to OBCPs (2 OPISS instances)
    - 1 dedicated to onboard file management
  - « Horizontal modularity »
    - Minimum interaction between components
    - Extensive use of « publish/ subscribe » messages



COMMUNICATION PROTOCOLS HANDLING

- Distributed OBSW architecture
  - Multiple configurations (model decoration)
  - No impact on components





SOMMAIRE

CONCLUSION

- AGATA development process
  - Model driven approach
    - Unique language and unique model ➔ UML (instead of several DSL)
    - Unique development environment ➔ Eclipse (all tools used are open source)
    - ➔ Y-shaped life cycle
  - Component based engineering
    - Best answer to increasing systems complexity
    - Relies largely on interface standardization
    - Normalization is required ➔ COrDeT
- System issues
  - Autonomy
    - Component-based architecture (reactive/deliberative decision process)
    - Short implementation loop (validation process nested in development process)
  - « Hardware independency »
    - Hierarchical architecture with abstraction of equipment units (monitors)
    - Communication layer generated from UML model (several execution platforms supported)
    - Dispatching of SW functions can be made via model decoration

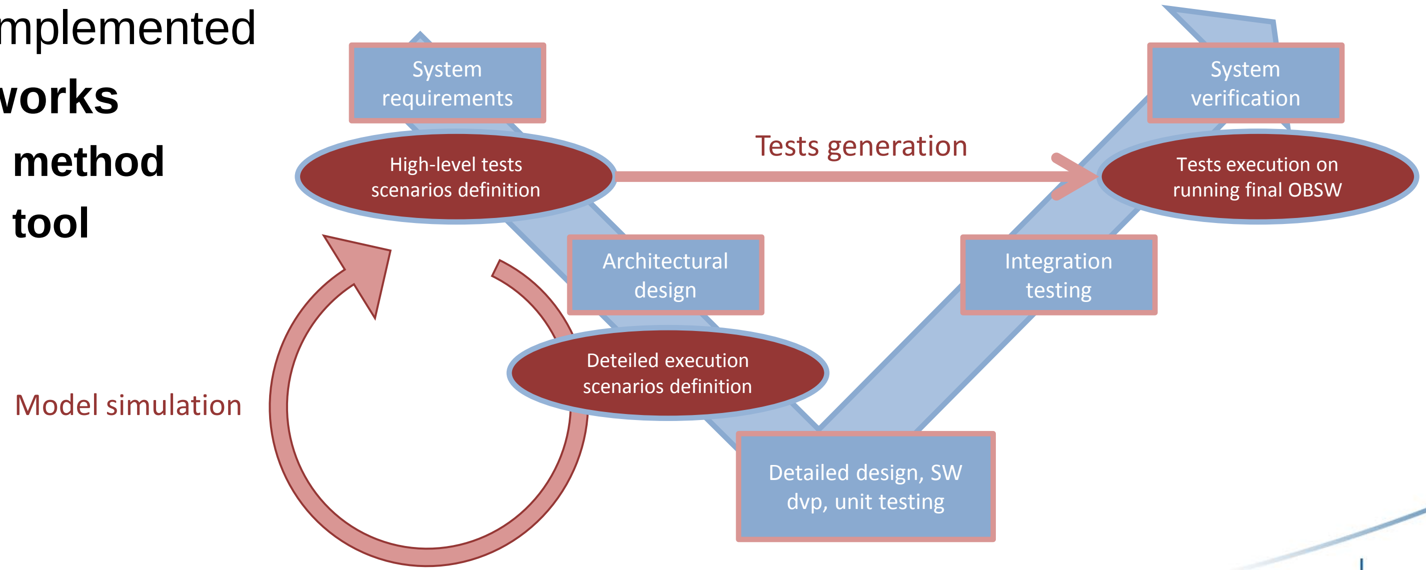
Thank you for your attention !

Questions ?

ANNEXES

UNIFIED MODEL-BASED VALIDATION

- Commonalities between validation tests generation and model simulation
  - Require UML specification of the OBSW
  - Tests scenarios described using activity diagrams
  - Existing – but independent – Topcased plugins
- Complementary approachs
  - Early model-debbuging using model simulation (validation of OBSW specification)
  - Intermediary model + code validation (“co-simulation”)
  - Validation of the final OBSW using tests generation
- Not yet implemented
- **Future works**
  - **Unified method**
  - **Unified tool**



UML GENERATORS PROJECT

- **Available shortly in PolarSys Eclipse Industrial Working Group**
- Based on Acceleo 3 (Obeo technology)
- Any generator which consumes or produces UML models
- The initial contribution provides five generators (4 developed for CNES)
  - UML2Java: converts Class and State diagrams into Java code
  - UML2C: converts Class, Activity and State diagrams into C code
  - C2UML: reverses C code into a UML model
  - UML2RTSJ (requires Autojava profil): converts Structure Composite, Class and State diagrams into Java code or RTSJ (Real Time Specification for Java) code
  - Java2UML: reverses Java code to a UML model

POLARSYS
Open Source Tools for Embedded Systems