AGENCE NATIONALE DE LA RECHERCHE ANR

SPaCIFY

MDE and Formal Methods for Embedded Systems Design

Alexandre CORTIER
Post-Doc CNES-IRIT
cortier@irit.fr

18-19 May 2010

Main concern : On-Board Flight Software

Satellite System architecture is specialized according to the satellite mission.

- Two main subsystems:
  - **payload**: application equipment (specific scientific instrumentation)
  - **platform**: flight software, communication devices, thermal regulation...

SPaCIFY exploratory project focuses on the flight software embedded in the satellite :

- hard real-time software
- ex: AOCS (Attitude and Orbit Control System), managing thermal regulation systems, monitoring systems status, managing on-board network, communicating with ground station.

Flight Software Engineering Process

Flight software is critical to the success of the mission:

- space industries and agencies worked on engineering processes
  - Goal: increase reliability
  - Ex: standards on software engineering and on product assurance (ECSS-Q-ST-80, ECSS-E-40)

BUT: These standards do not prescribe a specific process.

- They rather formalize documents, list requirements of the process and assign responsibilities to involved partners.

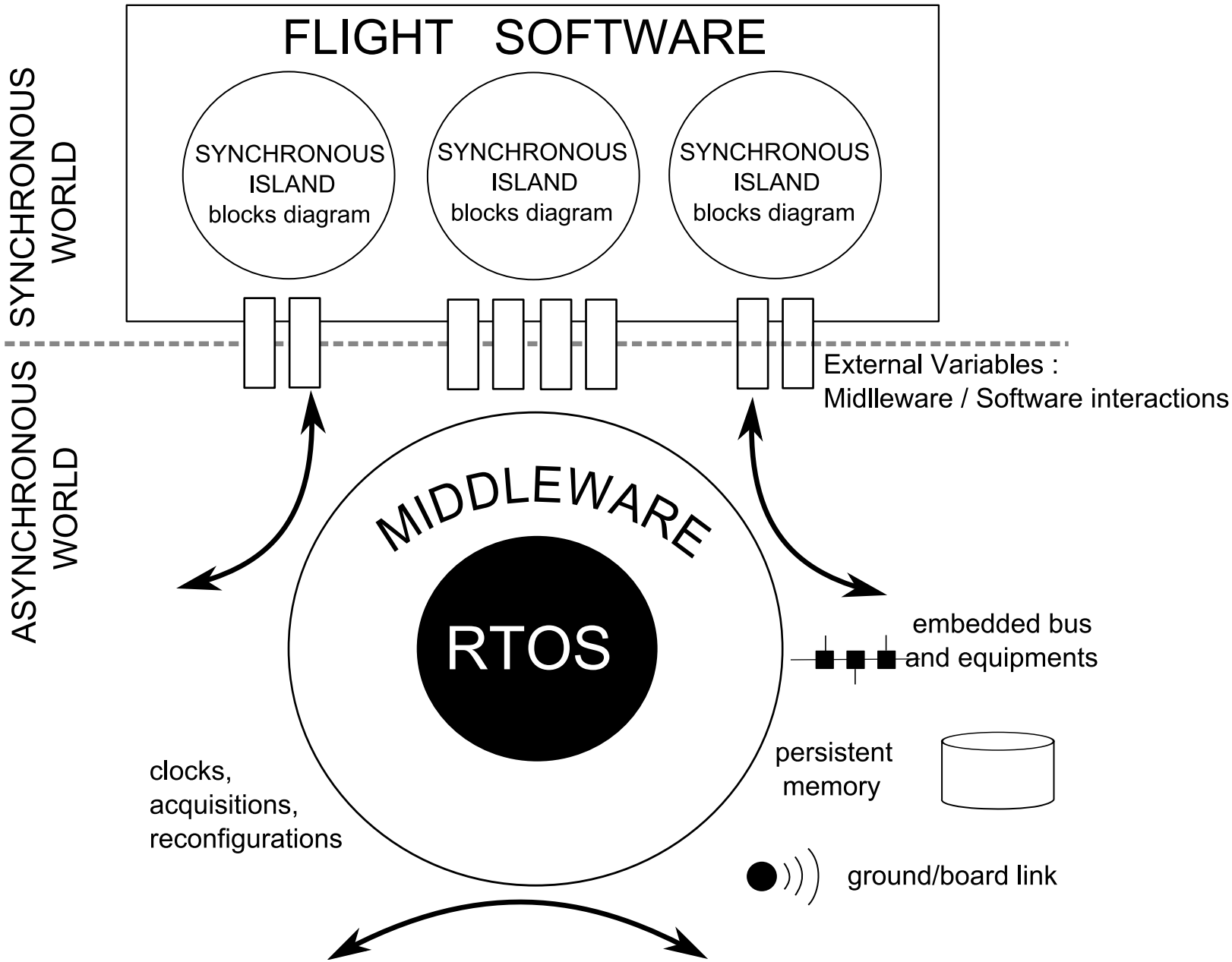
GALS Systems

Satellite management software:

- usually divided in parts that are quite autonomous one from the other
  - can share the same platform and resources
  - '*soft*' real-time constraints
  - asynchronous exchange
- Inside each of these parts, the subparts are most of the time strongly linked:
  - hard real-time constraints
  - synchronous exchange

This kind of systems is usually called **Globally Asynchronous, Locally Synchronous (GALS).**

GALS Systems



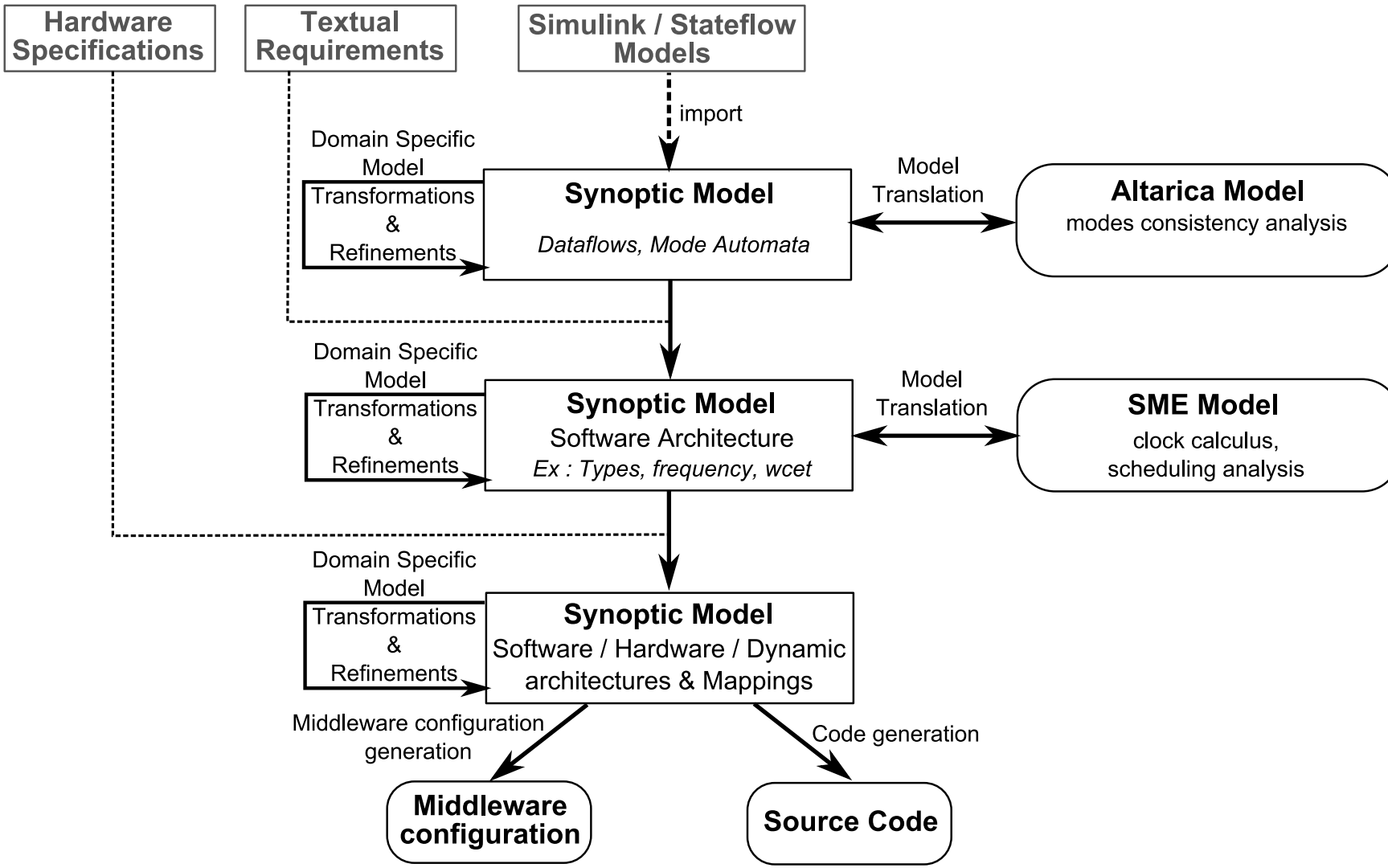
SPaCIFY Project objectives

The SPaCIFY ANR (French Research National Agency) exploratory project aims to define a design process and supporting tools for On-Board Flight Software based on:

- **Model-Driven Engineering (MDE)**
  - use models as a communication medium
  - to benefit from tools and techniques of the domain
- **Formal Methods**
  - multi-clock synchronous paradigm
  - model-checking
  - transformations verifications
- **Globally Asynchronous Locally Synchronous System (GALS) paradigm**
  - specification of the services of an executive platform
  - executive platform supporting distribution, partitioning and dynamic adaptation (middleware)

The ssociated tools are built upon the **Topcased toolkit**. (*The Open-Source Toolkit for Critical Systems*)

SPaCIFY Process



Synoptic

Synoptic is the **core language** of the SPaCIFY process:

- **a graphical and textual DSML**
- **provides high-level constructions** to handle
  - multi-layers description (various modeling aspect)
  - various granularity levels (iterative and refinement development)
  - modular approach
- **based on a synchronous semantics**
  - formal and deterministic analysis and verification
  - refinement proof
  - transformation proof

Synoptic : multi-layers system specification

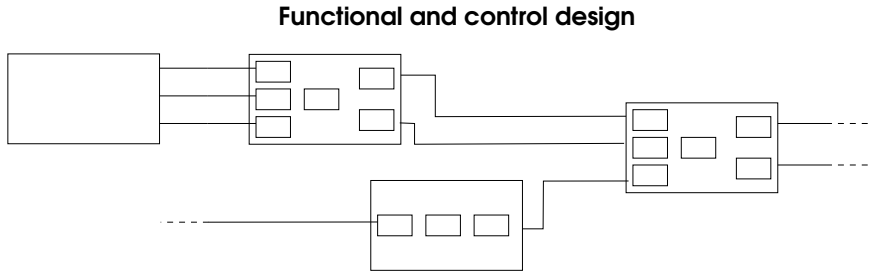
Synoptic is not fundamentally a new language but an integration of different sources and concepts.

Synoptic is inspired by several approaches :

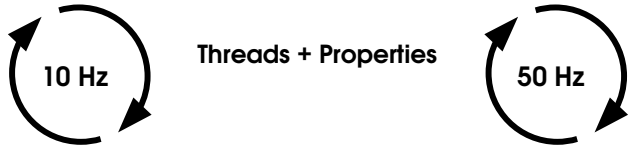
- **Geneauto**: safe subset of the Simulink/Stateflow modelling language used for the development of certified safety critical embedded real time systems
  - structural feature: Dataflow models ("Blocks Diagrams")
  - behavioral feature: Control Flow models ("Finite States Machines")
  - real-time constraints: clock properties
- **AADL**: *Architecture Analysis & Design Language* (formerly Avionics Architecture Description Language)
  - Threads description
  - platform aspects ("components view")
  - mappings: which component execute which functional blocks ?
- **Components Models**: CCM, Fractal
  - Synoptic components : need to be improved...

Synoptic : multi-layers system specification

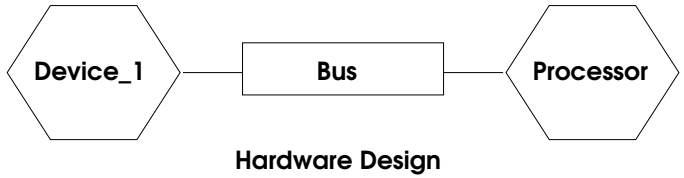
Software architecture



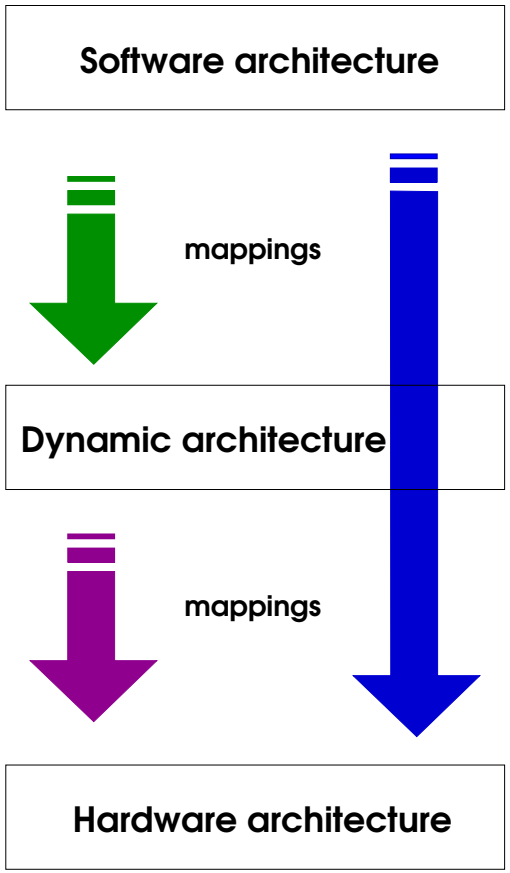
Dynamic architecture

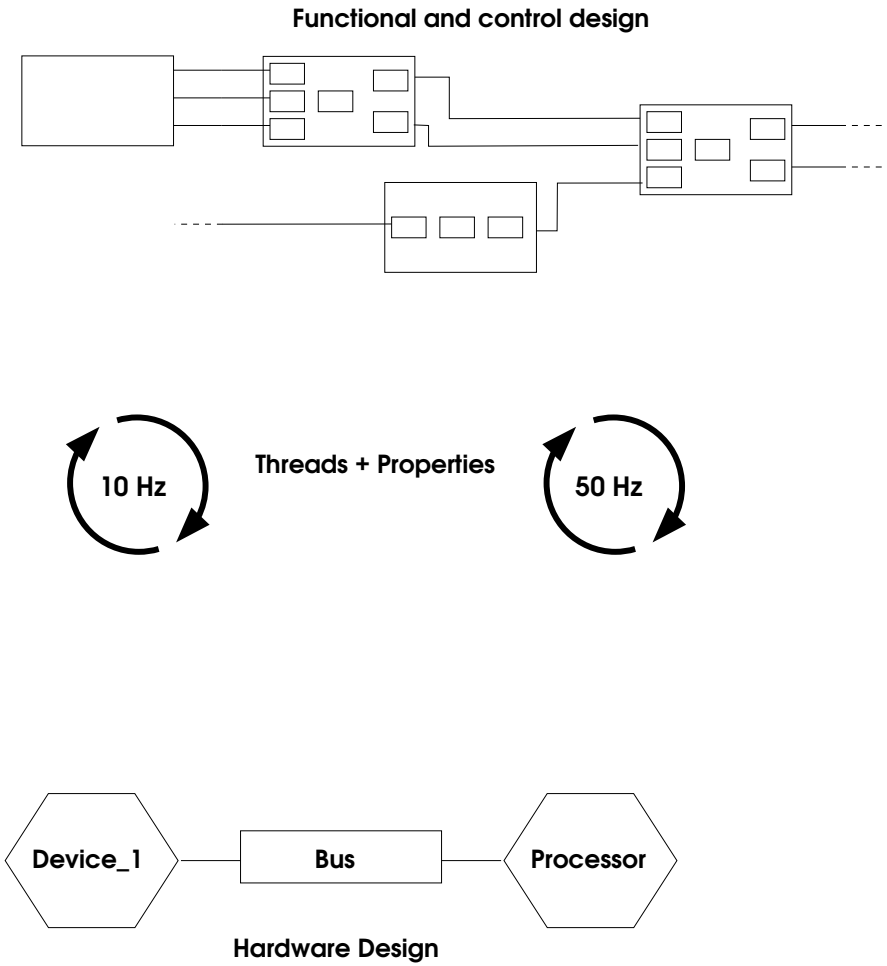


Hardware architecture

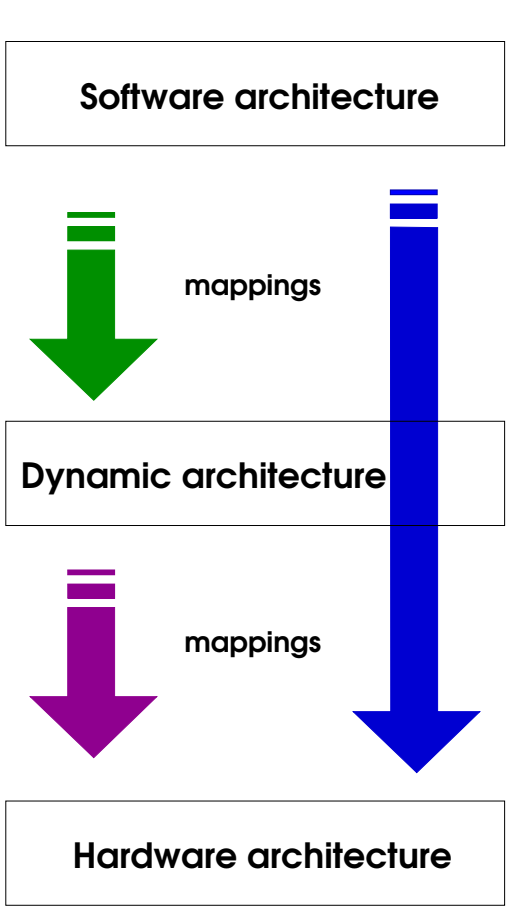


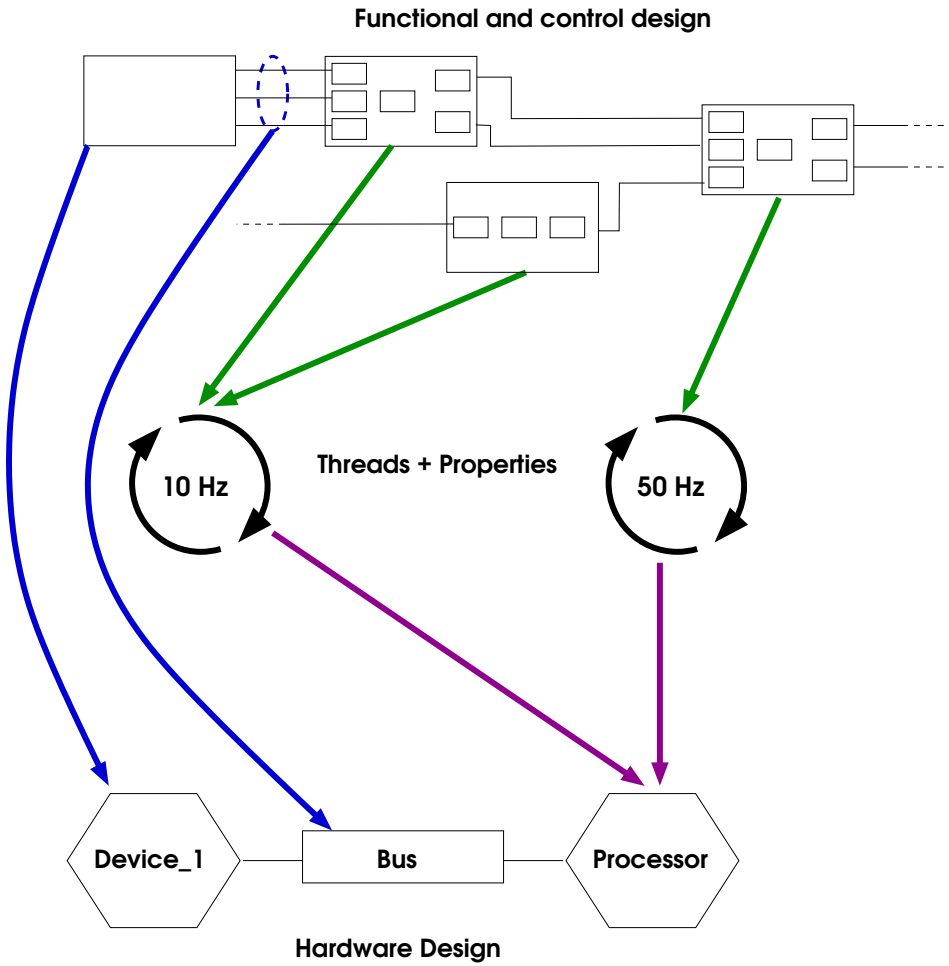
Synoptic : multi-layers system specification





Synoptic : multi-layers system specification

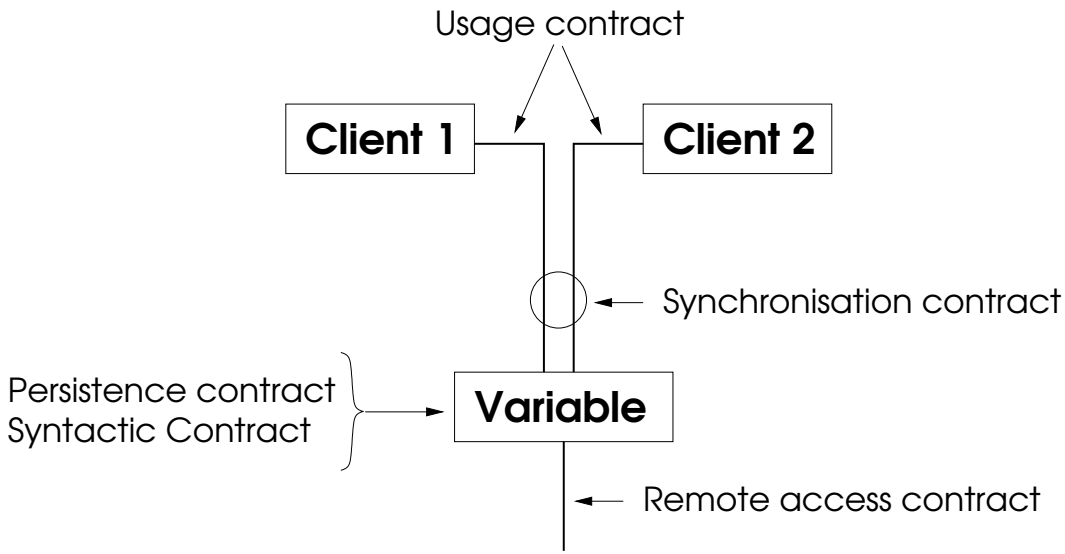




Synoptic/MW : external variables

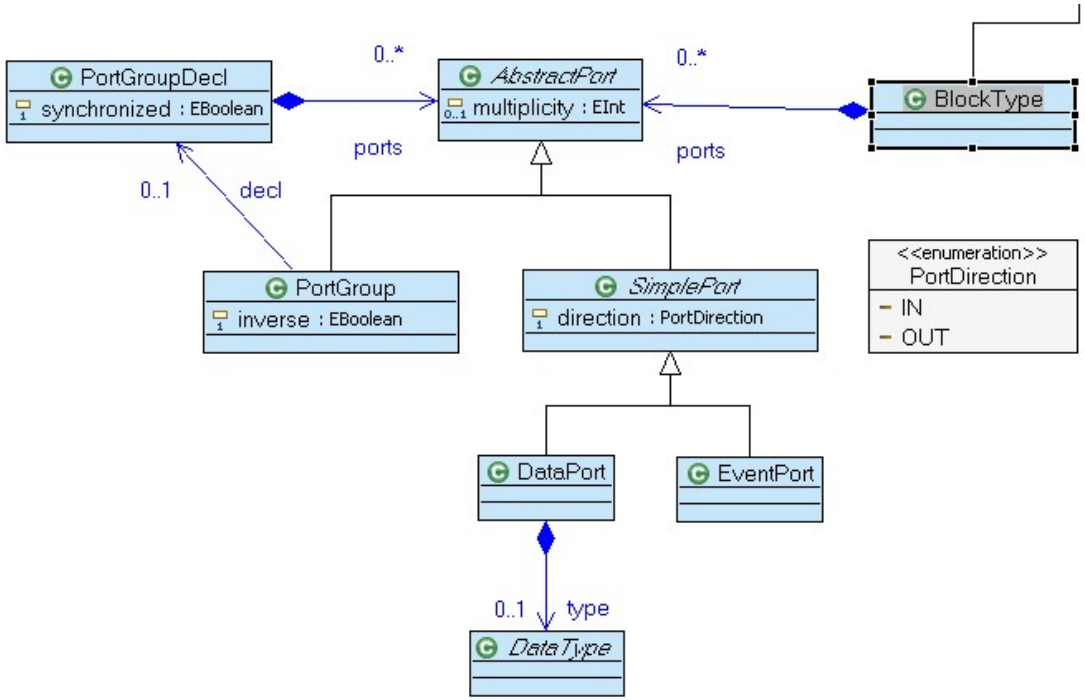
The Middleware has to abstract the asynchronous behavior of the system (bufferisation,...).
Interactions between MW and Synoptic models are handled using **external variables** concept.

- external variables = sources / sinks of signals
- external variables types: constants, TM, TC, global variables...
- external variables contracts



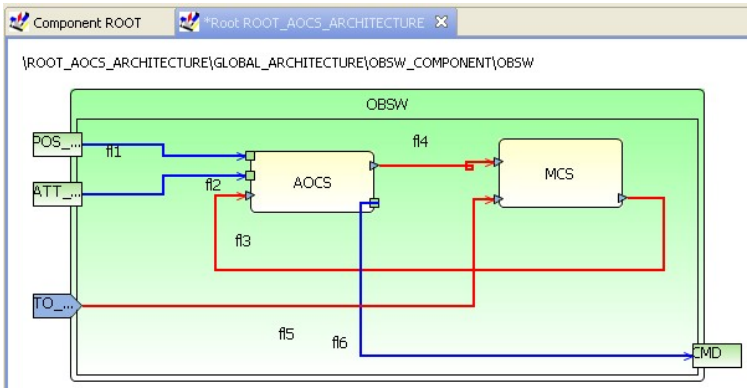
External variables and associated contracts are used to configure the MW.

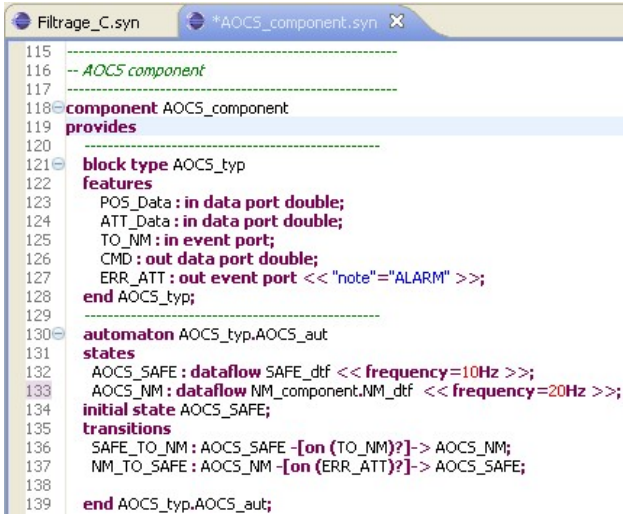
Tools chain : Meta-Model of Synoptic DSML



- *Meta-model of Synoptic describe using the formalism ECore*
- *ECore* = metamodeling architecture in the Eclipse Modeling Framework (EMF)
  - *more or less aligned on OMG's metamodeling architecture MOF (Meta-Object Facility)*

Tools chain : Textual & Graphical editor





- **Graphical editor** prototyp (Anyware Technologies) :
  - based on the **EMF/Topcased** framework
- **Textual syntax** defined with **TCS** (Textual Concrete Syntax)
  - TCS is a DSL defined with a KM3 metamodel
  - can be used to:
    - parse text-to-model
    - serialize model-to-text
  - performed with *a single (bidirectional) specification*

Horizontal/Vertical Transformations

1. **Translation Transformation**: Synoptic → Altarica
   - verifying properties such as *coherence of the modes*
     - model-checking
   - Ex : '*if component X is in mode m1, then component Y is in mode m2 or can eventually move into mode m2*'
2. **Translation Transformation**: Synoptic → SME
   - SME : meta-model of Signal (synchronous language)
   - temporal analysis (clocks), scheduling analysis, code generation
   - using the Polychrony platform
   - translation describes using the Kermeta language
3. **Domain Specific Transformations (in progress)**
   - Ex : Model organization, automata elicitation
   - Ex : Software function splitting and mapping to threads from RTOS

Case Study: Thales Alenia Space

TAS uses a model based approach relying on the components model lifgtweight CCM to handle the deployment of the system (except for functional implementation of the components)

- **Use Case Characteristics:**
  - Modeling the OBSW (central flight software)
  - Sub-systems : Battery Management, Thermal regulation management, AOCS, global FDIR, actuators, sensors
- **Goals:** prove the utility of using Synoptic models as a unifying and unique design
  - to **model the structural aspects** of the software and to deduce the CCM model and the configuration of the middleware by model transformation
  - to **model the behavioral aspects** and to automatically generate the implementation of CCM components using the Polychrony platform
- **Results:**
  - code generation has been proved to be feasible

Case Study: EADS Astrium

Astrium experiments technologies with a real industrial use case:

Satellites flying in formation

- **Characteristics:**
  - Coordination of 2 satellites (1 master and 1 slave)
  - FDIR (Failure Detection Isolation and Recovery) more complex
- **Goals:** to cover a horizontal slice of the SPaCIFY engineering process
  - Evaluation of the Synoptic modeling language for early system engineering phases
  - Evaluation of the Altarica-based model-checker, especially with respect to its scalability
    - Validation of the Control/Command strategy
- **Results:**
  - the model contains 49 automata
  - model has been translated in Altarica
  - invariants (coherence of modes) have been verified using ARC and MEC (Labri model-checkers)

Case Study: CNES

- **Characteristics:**
  - modeling of the Control/Command part of a Payload Manager
  - sub-systems: House-Keeping, Routing, Filtering
- **Goals:**
  - test the expressivity of the Synoptic language
  - test if the components of the middleware can be modeled in Synoptic
  - better understand the interconnection synchronous islands/middleware (external variables)
- **Results:**
  - asynchronous exchanges can be modeled in Synoptic (using a 'fresh boolean data')
  - filtering component can be modeled in a generic way

Conclusions

SPaCIFY project defines a design process for on-board flight software based on:

- Model Driven Engineering
- GALS
- Formal Methods: Synoptic equipped with a formal synchronous semantics

A prototype tool chain based on the Eclipse Modeling Framework:

- textual and graphical editor
- translation transformations into Altarica and SME (Kermeta)
- code generation using the Polychrony platform
- verification of the coherence of modes using ARC and MEC (model-checker Altarica)
- OCL constraints have been encoded to check structural constraints on models

Works in progress

The case studies help to highlight some improvements to do on the Synoptic language:

- automata in Synoptic
- adding the notion of partition to capture IMA concept
- external variables

Domain Specific Transformations :

- formal semantics of the language has been encoded in a typed sets Theory using the B Method
  - will be helpful to validate the existing transformation Synotic to SME models
  - will be used to the formalization of domain specific transformations
    - refinement transformations
    - model reorganization
    - automatic mapping of functional blocks on threads

Perspectives

- Implementation of a **clock calculus** for Synoptic
- Extend Synoptic with the formal concept of **contracts (assume/guarantee)**
  - refinements
- Improve code generation (modularity)
- **Formal correctness proof and subsequent certification** of a code generator
  - under way in the GeneAuto project

Thank You.