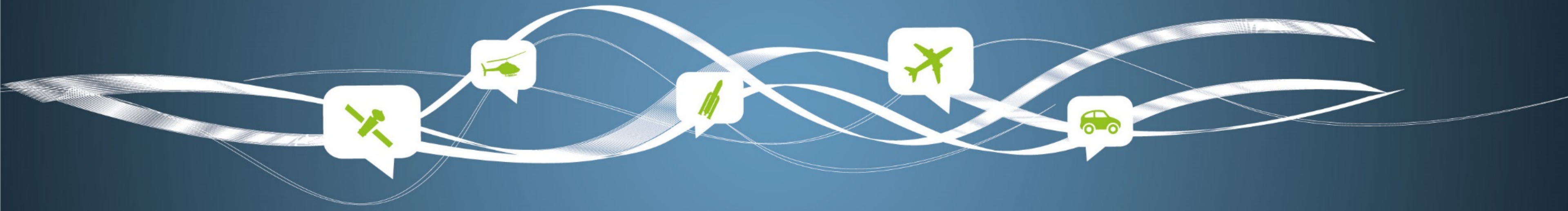
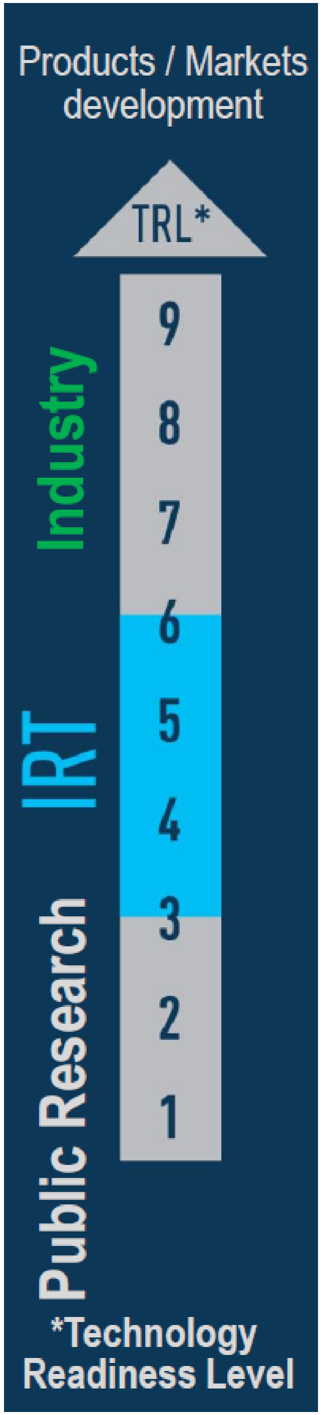
IRT
SAINT EXUPERY

Model Driven Engineering with Capella and AADL

Bassem Ouni, Pierre Gaufillet, Eric Jenn, Jérôme Hugues

8th European Congress
ERTS² 2016
EMBEDDED REAL TIME SOFTWARE AND SYSTEMS
27-29 JANUARY 2016 / TOULOUSE, FRANCE
CENTRE DE CONGRÈS PIERRE BAUDIS
www.erts2016.org





Products / Markets development

TRL*

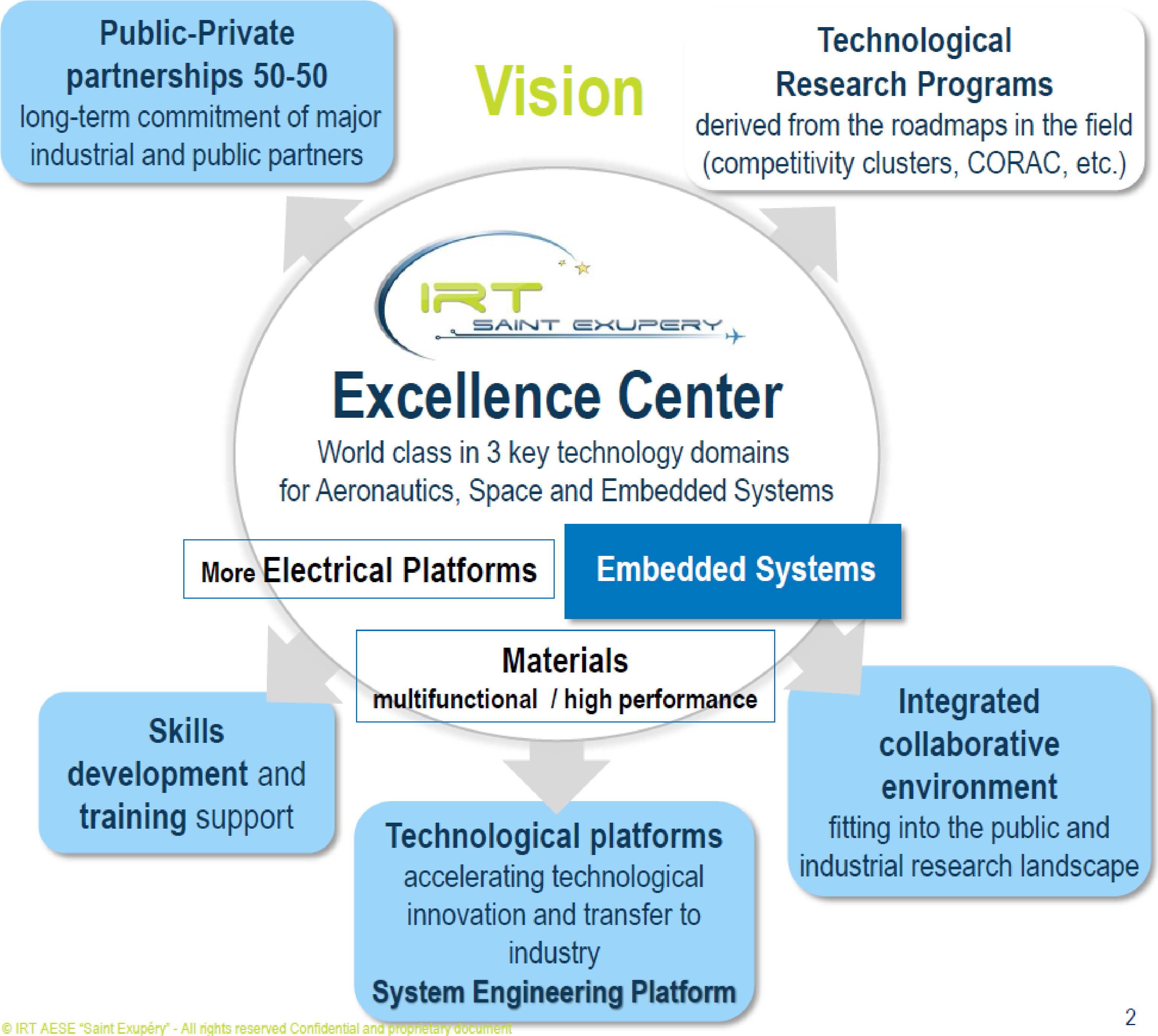
9
8
7
6
5
4
3
2
1

Industry

IRT

Public Research

*Technology Readiness Level



Vision

Public-Private partnerships 50-50
long-term commitment of major industrial and public partners

Technological Research Programs
derived from the roadmaps in the field (competitivity clusters, CORAC, etc.)

Excellence Center

World class in 3 key technology domains
for Aeronautics, Space and Embedded Systems

More **Electrical Platforms**

Embedded Systems

Materials
multifunctional / high performance

Skills development and **training** support

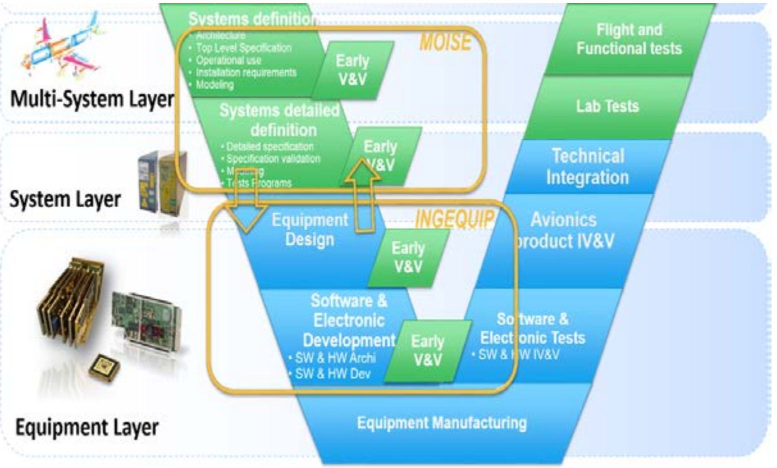
Technological platforms
accelerating technological innovation and transfer to industry
System Engineering Platform

Integrated collaborative environment
fitting into the public and industrial research landscape

Introduction

- Projects INGEQUIP and MOISE at IRT Saint Exupéry

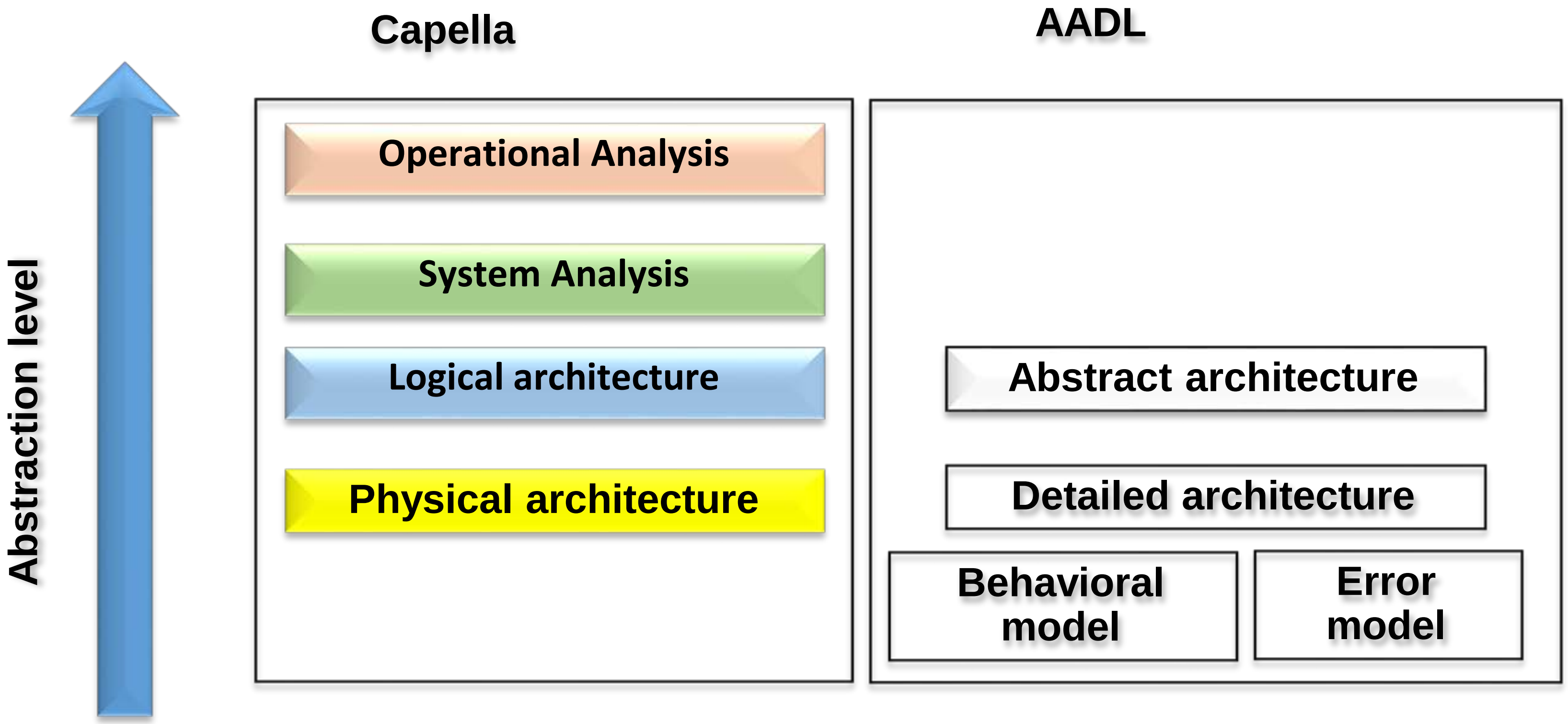
- INGEQUIP
  - Co-development agility to assist Equipment design
  - An activity focused on HW-SW Co-design
  - Applies methods and tool on large demonstrator
    - Mobile vehicle or Rover *TwIRTee*



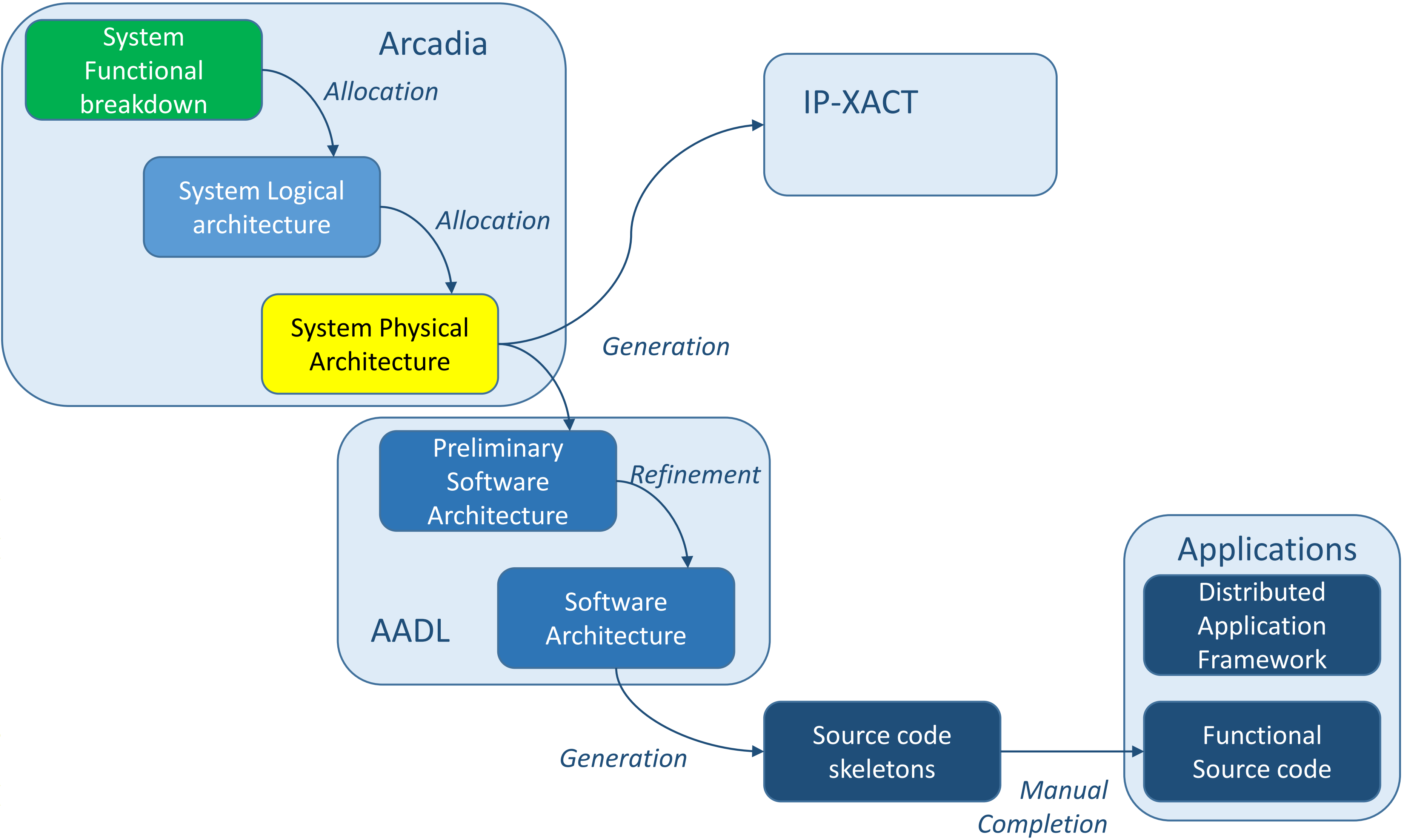
- MOISE
  - *MOdels and Information Sharing for System engineering in Extended enterprise*
  - From document-based specification process to model-driven specification process
  - Ensure multi-view system design and verification in a heterogeneous context
  - Collaborative engineering processes and methods in a regulation ("certification")

Why Capella to AADL?

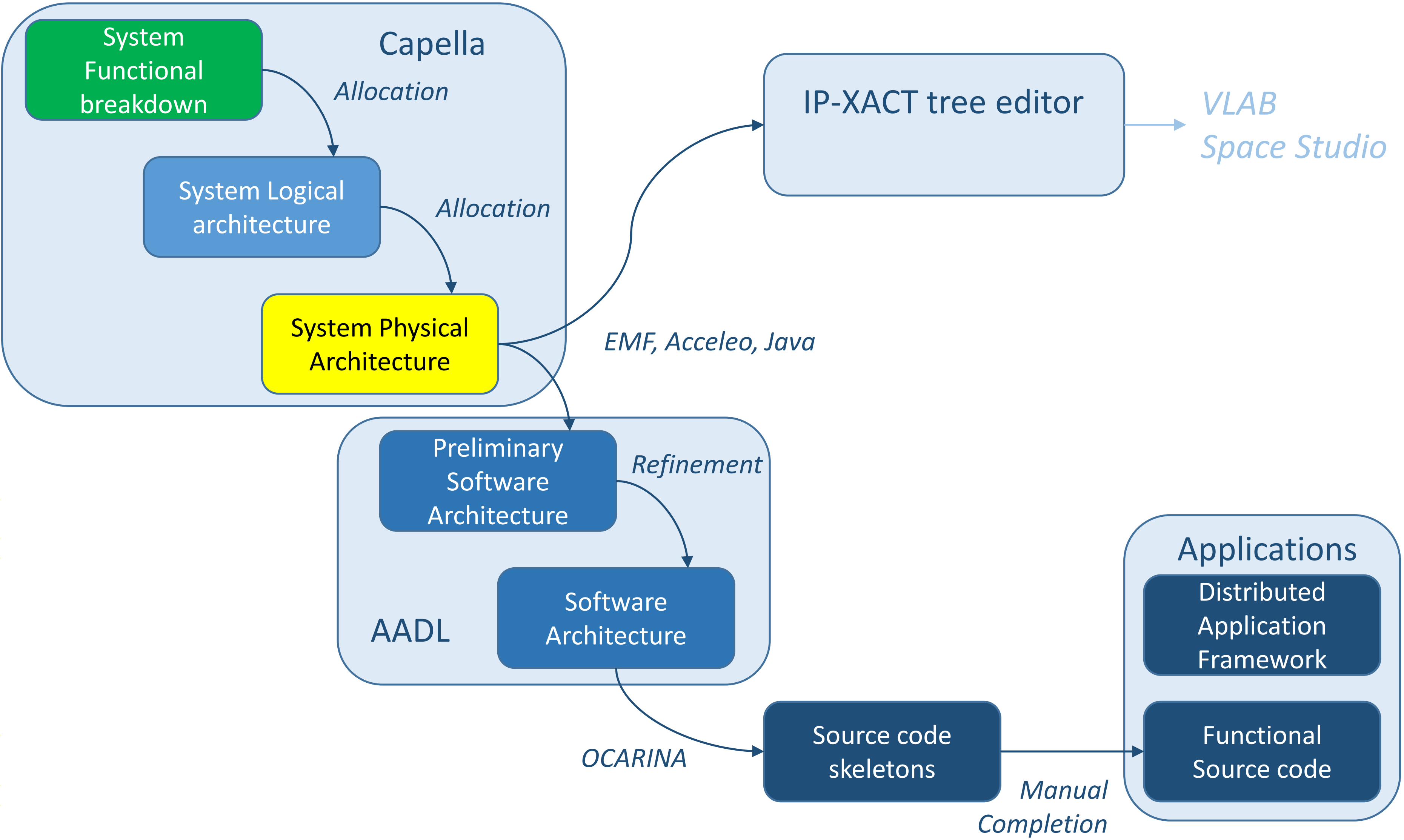
- Capella is positioned on the most abstract part of the system development while AADL doesn't offer operational or system analysis
- AADL goes further than Capella in terms of detailed architecture with explicit hardware components – *device, processor, memory, bus* – and explicit software components – *process, thread, subprogram*.



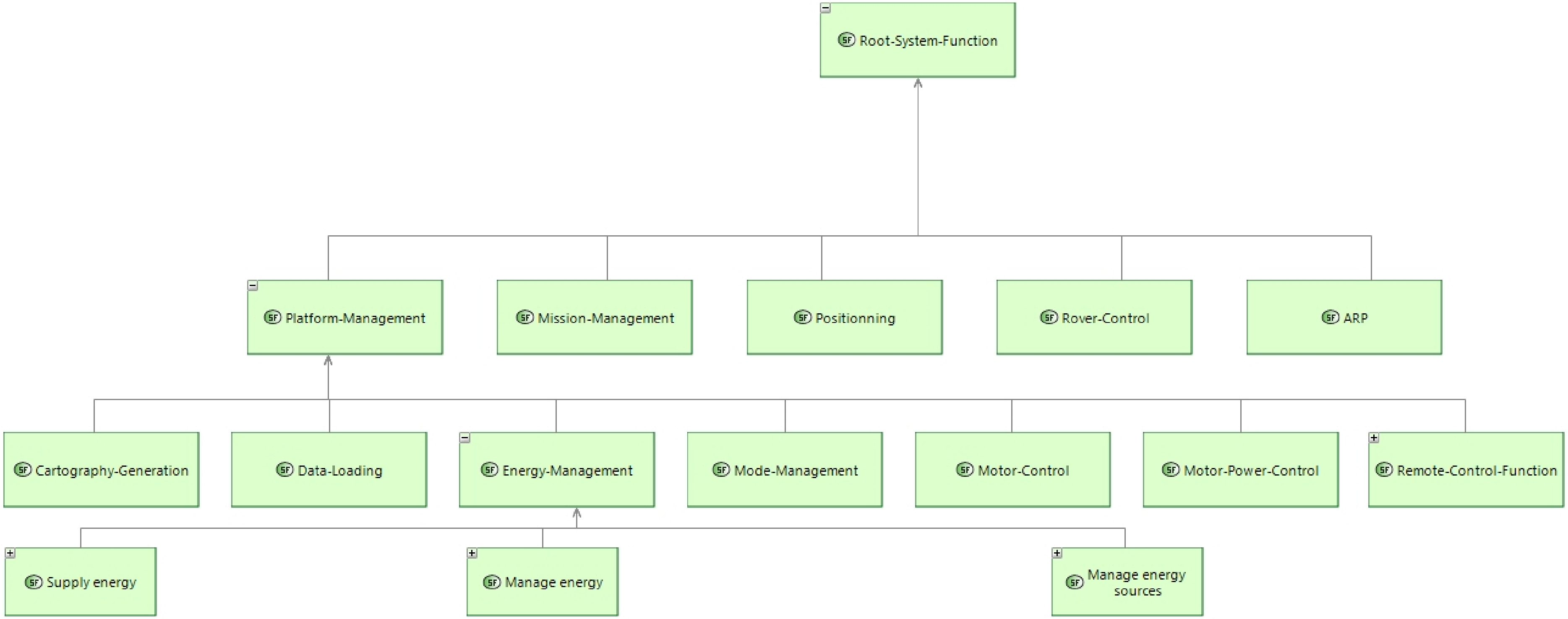
Target chain



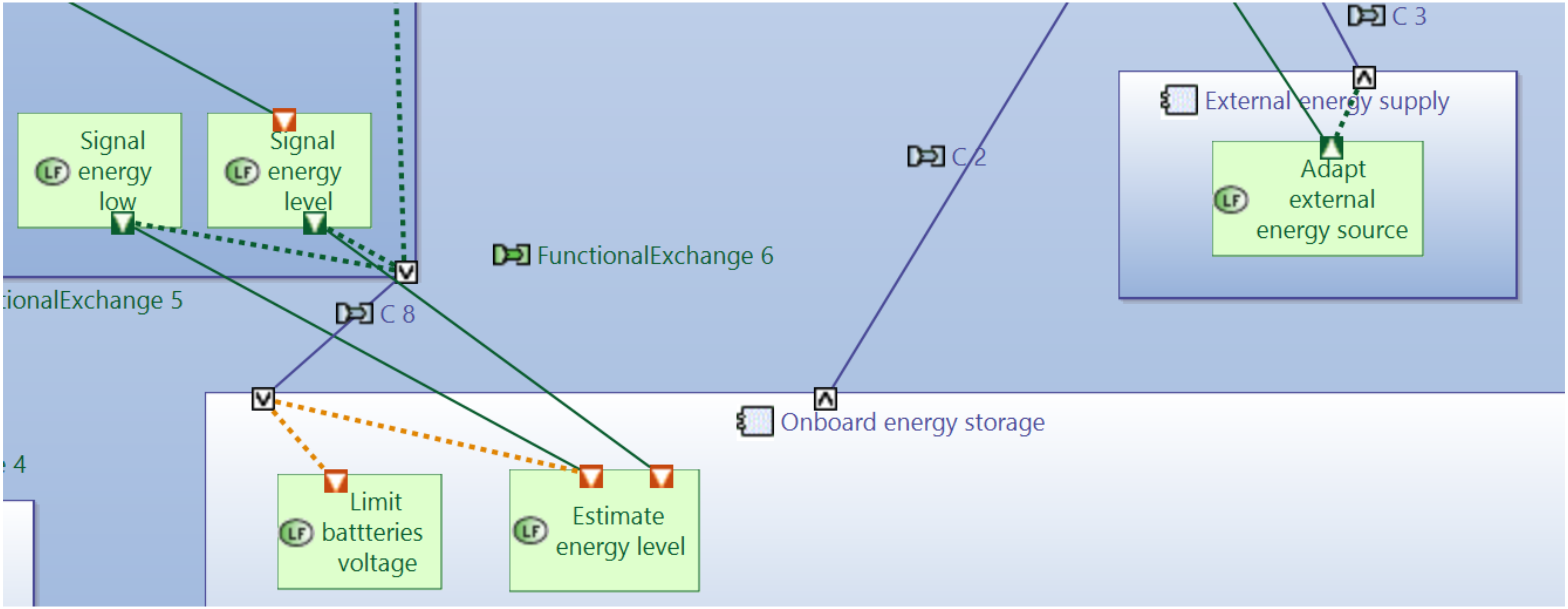
Target chain



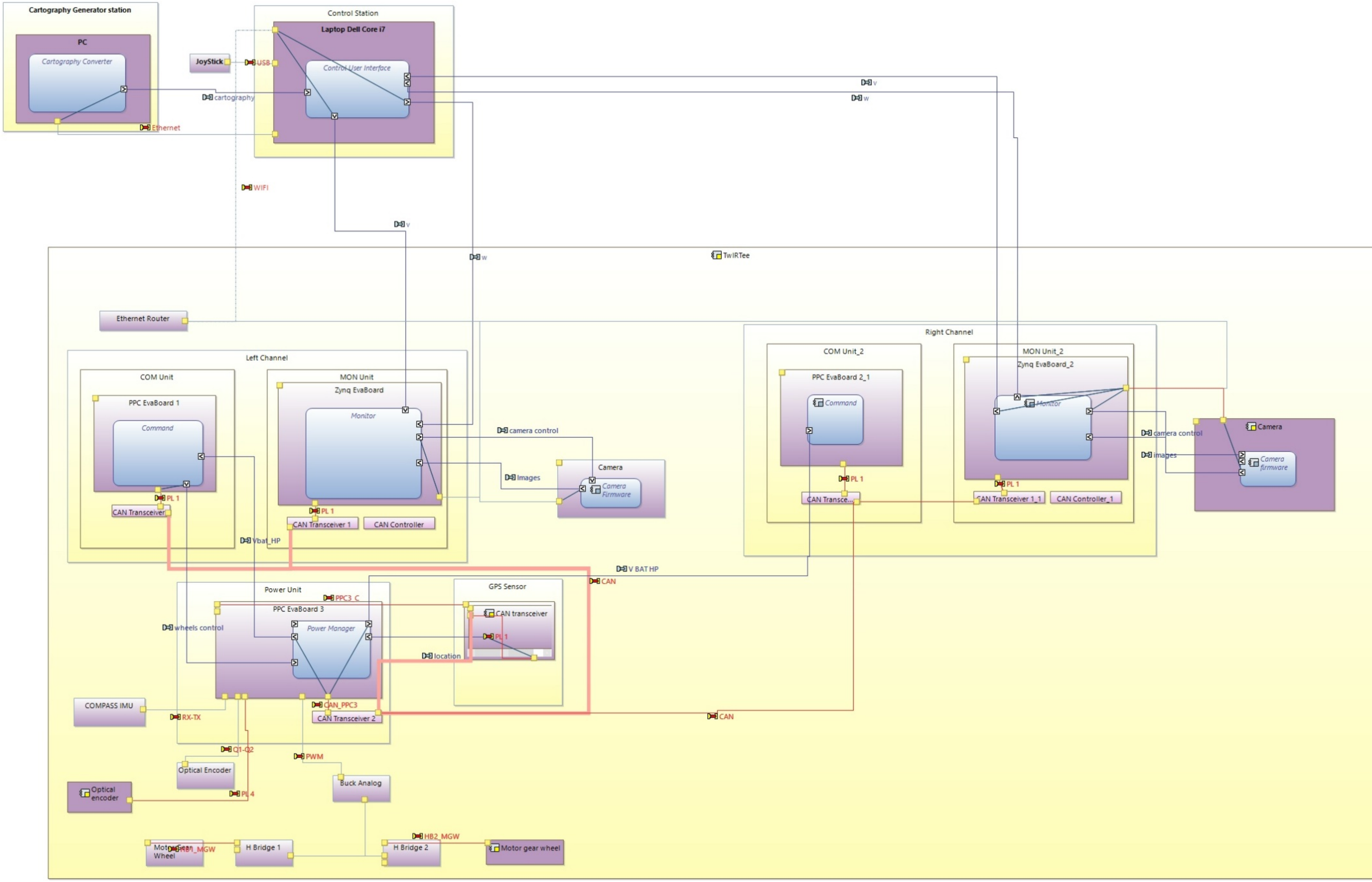
TwIRTee Functional Breakdown



TwIRTee Logical Architecture



TwIRTee Physical Architecture



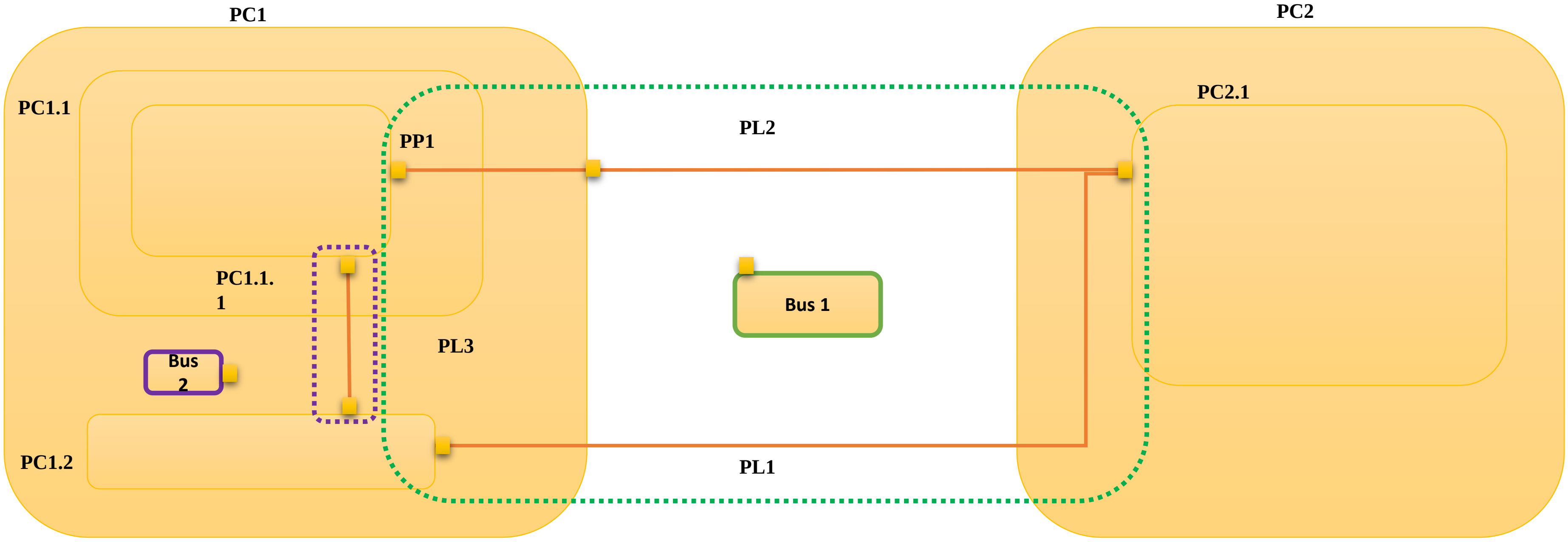
System architecture with Capella

- Modelling conventions
  - Physical architecture breakdown stops when a leaf is allocated to hardware – device - or software – application.
  - Flows are neither typed nor constrained
  - Interconnected ports implicitly define a bus
  - Software is allocated to CPUs and logical flows to physical flows
  - System functions are not used

Capella to AADL mapping

Capella	Condition	AADL
PhysicalComponent	nature =PhysicaComponentNature::**NODE** and kind=PhysicalComponentKind::**SOFTWARE_EXECUTION_UNIT**	Processor
PhysicalComponent	nature =PhysicaComponentNature::**NODE** and kind=PhysicalComponentKind::**HARDWARE**	Device
PhysicalComponent	nature =PhysicaComponentNature::**BEHAVIOR** and kind=PhysicalComponentKind::**SOFTWARE_APPLICATION**	Process
PhysicalComponent	other than above	System
PhysicalPort	See Bus extraction algorithm	Requires bus access
PhysicalLink		Bus access connection + Bus
ComponentPort	kind=ComponentPortKind::**FLOW** and direction=OrientationPortKind::[**IN|OUT|INOUT**] See Feature connection extraction algorithm	Feature
ComponentExchange	See Feature connection extraction algorithm	Feature connection
ComponentPortAllocation	The bus type has to be precised on at least one Physical Link using a Physical Link Category	Actual_connection_binding property on bus
C case + container hierarchy	Components of Case C have to be contained by a component of case A	Actual_processor_binding property
Other metaclasses		N/A

Extraction of Bus



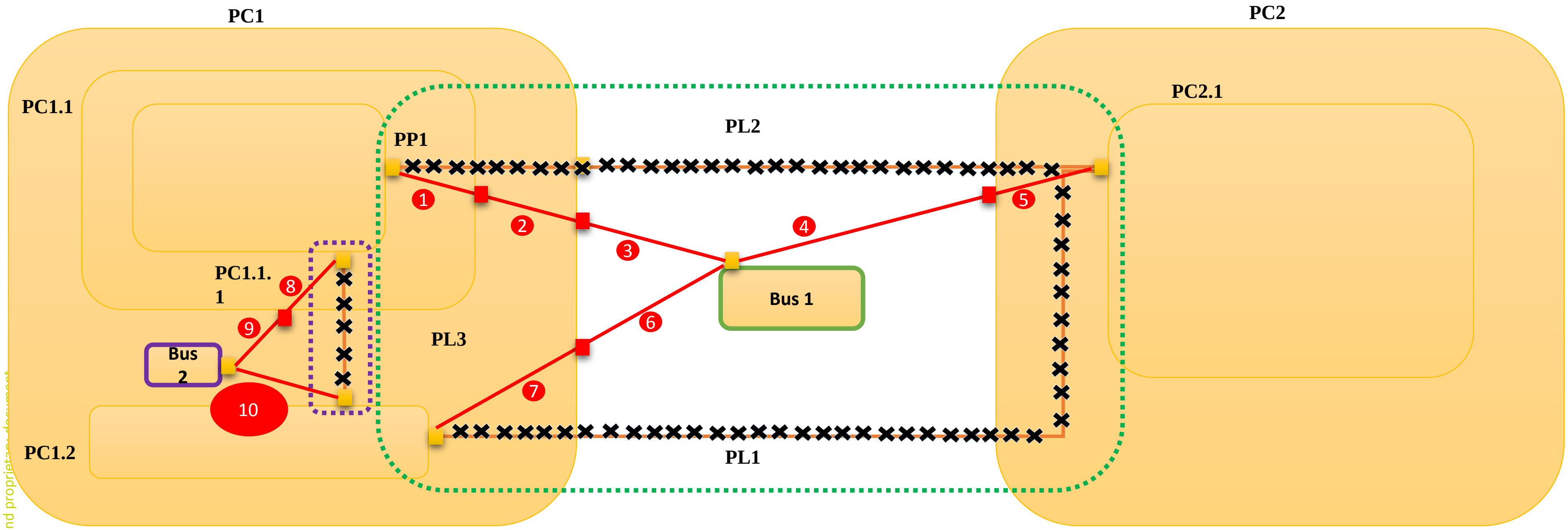
-Gathering groups of physical links which are connected to each other.

-Each group will be mapped in a physical bus (Physical component with Bus port).

-Connected Physical Links are translated into requires access connectors to a bus through intermediate ports.

PC: Physical component
PL: Physical link
PP: Physical port

Extraction of Bus



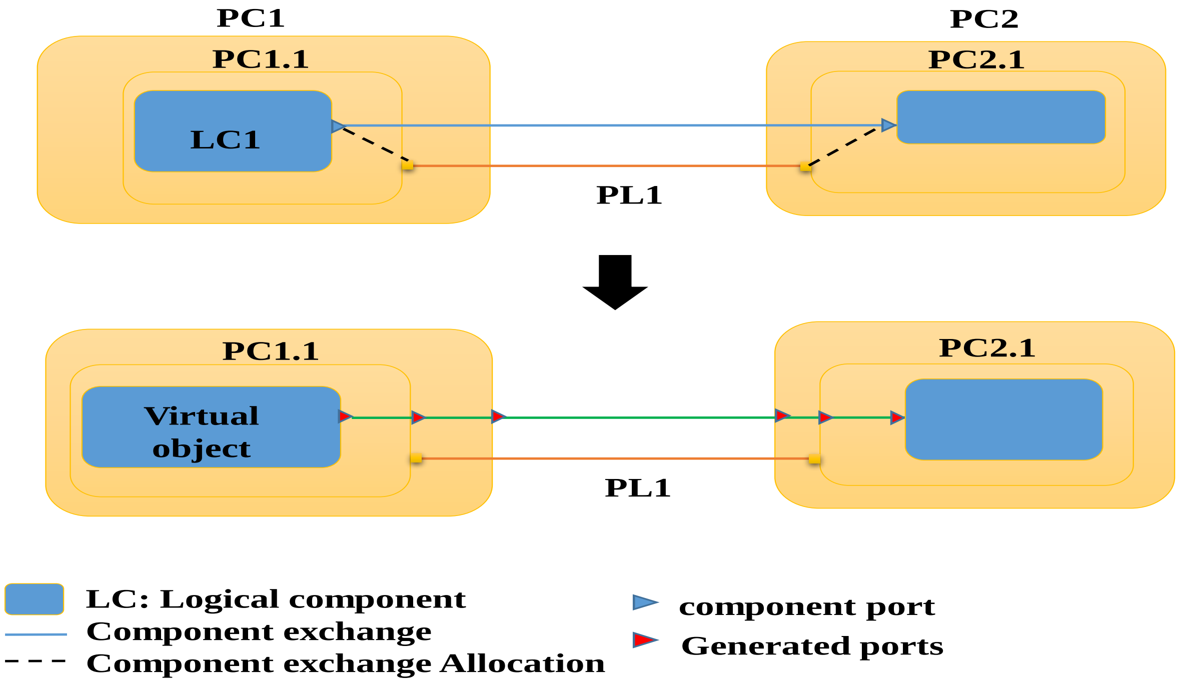
-Considering the architecture as a tree, we first search the external ports (Last level of the tree).

-Bus ports are connected to external ports.

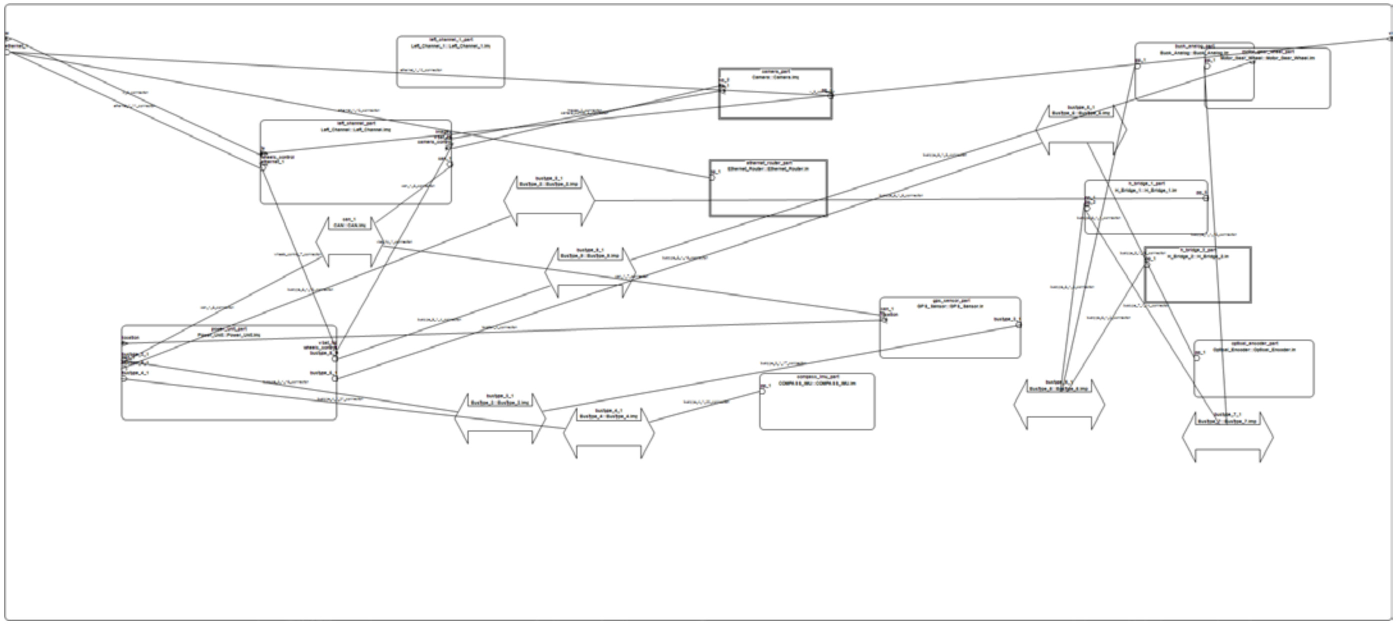
-Synthetic links and synthetic ports are generated=> mapped to AADL connections and requirements of bus accesses.

Extraction of logical connections

- Components exchanges are translated into oriented abstract flows through intermediate ports.



Preliminary Software Architecture with AADL

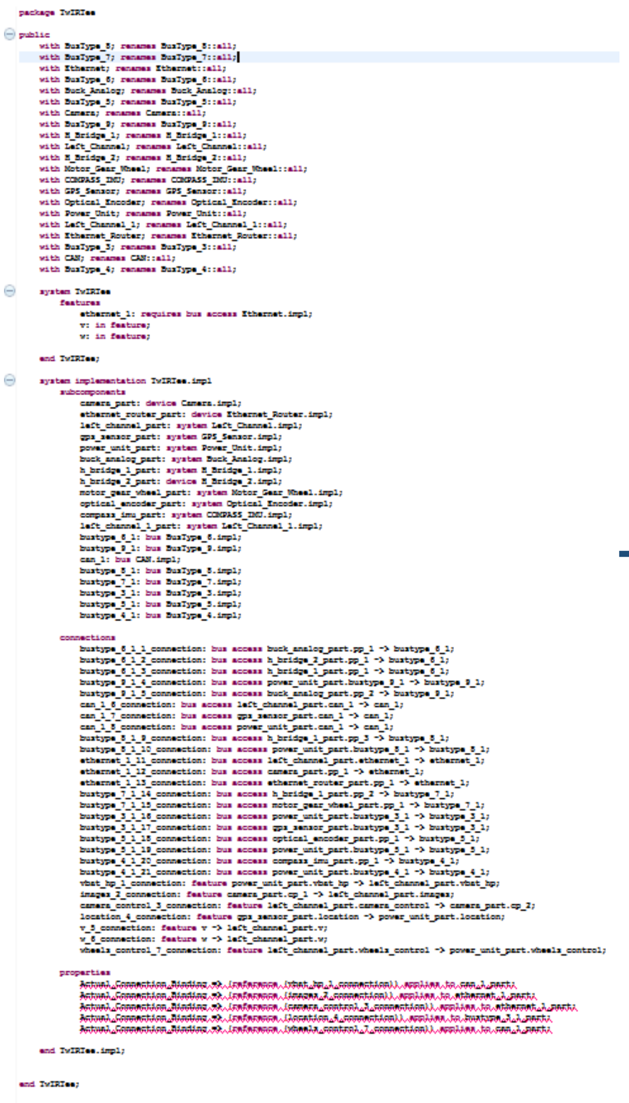

left_channel_1_part
Left_Channel_1::Left_Channel_1.imp
camera_part
Camera::Camera.imp
buck_analog_part
Buck_Analog::Buck_Analog.im
motor_gear_wheel_part
Motor_Gear_Wheel::Motor_Gear_Wheel.im
left_channel_part
Left_Channel::Left_Channel.imp
ethernet_router_part
Ethernet_Router::Ethernet_Router.im
h_bridge_1_part
H_Bridge_1::H_Bridge_1.im
h_bridge_2_part
H_Bridge_2::H_Bridge_2.im
can_1
CAN::CAN.imp
gps_sensor_part
GPS_Sensor::GPS_Sensor.im
power_unit_part
Power_Unit::Power_Unit.imp
compass_imu_part
COMPASS_IMU::COMPASS_IMU.im
optical_encoder_part
Optical_Encoder::Optical_Encoder.im
bustype_7_1
BusType_7::BusType_7.imp

- Fully generated from Capella
- Read-only artefact – Regeneration is straight-forward

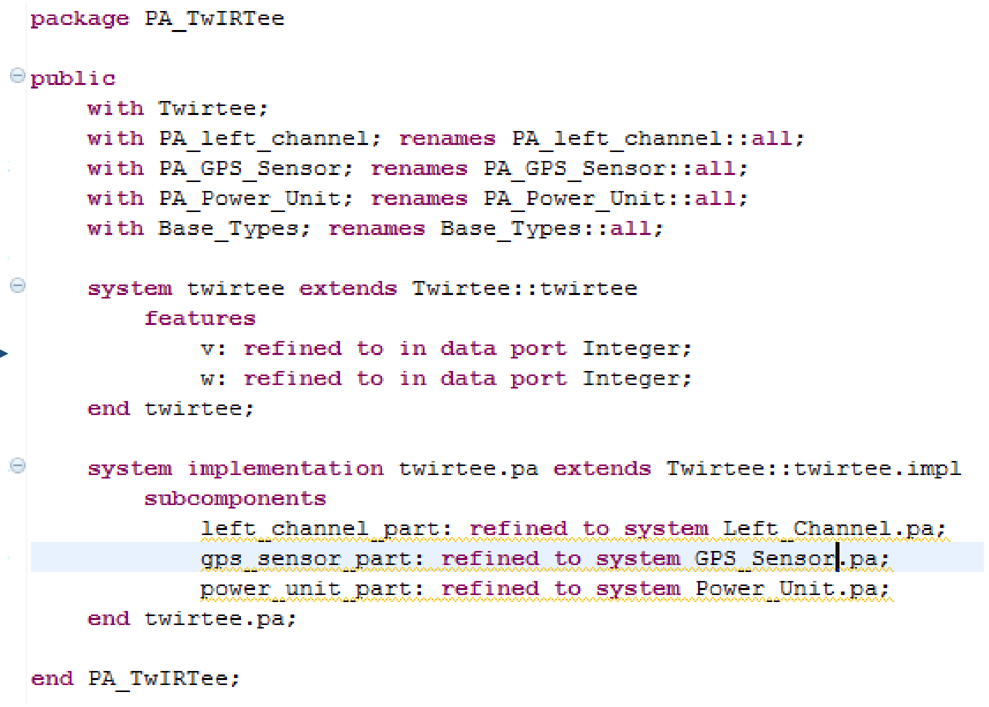
Software Architecture with AADL

- Inherits from the preliminary architecture
- Refined manually by adding
  - Internal application objects: tasks, buffers, semaphores, etc.
  - Typed flows
  - Static software architecture: application subsets, API
  - Real Time constraints
  - Abstract behaviour
- Impacts of modification in the preliminary architecture are easily identified thanks to inheritance and refinement relationships.
- Can be used for formal behavior verification

Software Architecture with AADL

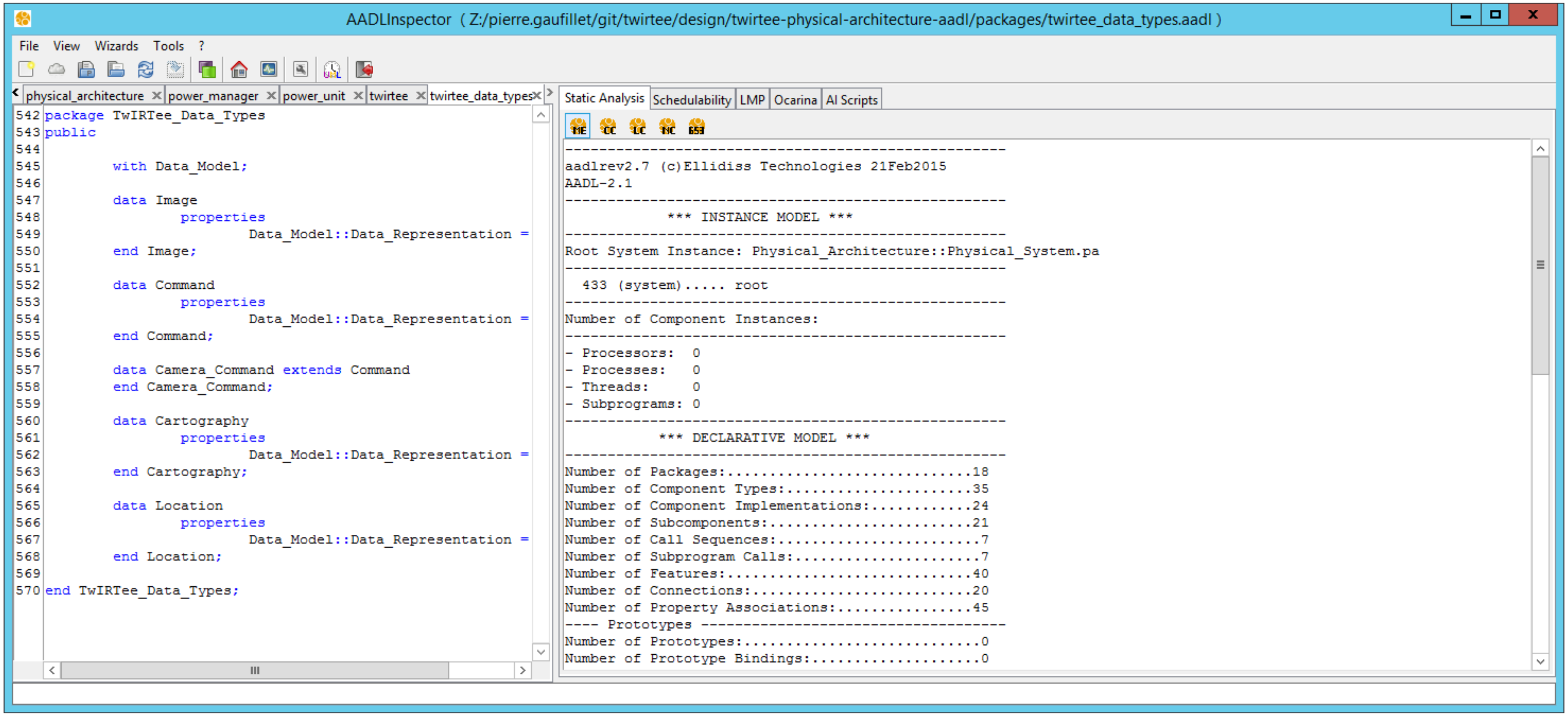


```
package TwIRTee

public
    with BusType_8; renames BusType_8::all;
    with BusType_7; renames BusType_7::all;
    with Ethernet; renames Ethernet::all;
    with BusType_6; renames BusType_6::all;
    with Buck_Analog; renames Buck_Analog::all;
    with BusType_5; renames BusType_5::all;
    with Camera; renames Camera::all;
    with BusType_9; renames BusType_9::all;
    with H_Bridge_1; renames H_Bridge_1::all;
    with Left_Channel; renames Left_Channel::all;
    with H_Bridge_2; renames H_Bridge_2::all;
    with Motor_Gear_Wheel; renames Motor_Gear_Wheel::all;
    with COMPASS_IMU; renames COMPASS_IMU::all;
    with GPS_Sensor; renames GPS_Sensor::all;
    with Optical_Encoder; renames Optical_Encoder::all;
    with Power_Unit; renames Power_Unit::all;
    with Left_Channel_1; renames Left_Channel_1::all;
    with Ethernet_Router; renames Ethernet_Router::all;
    with BusType_3; renames BusType_3::all;
    with CAN; renames CAN::all;
    with BusType_4; renames BusType_4::all;

    system TwIRTee
        features
            ethernet_1: requires bus access Ethernet.impl;
            v: in feature;
            w: in feature;

    end TwIRTee;

    system implementation TwIRTee.impl
        subcomponents
            camera_part: device Camera.impl;
            ethernet_router_part: device Ethernet_Router.impl;
            left_channel_part: system Left_Channel.impl;
            gps_sensor_part: system GPS_Sensor.impl;
            power_unit_part: system Power_Unit.impl;
            buck_analog_part: system Buck_Analog.impl;
            h_bridge_1_part: system H_Bridge_1.impl;
            h_bridge_2_part: device H_Bridge_2.impl;
            motor_gear_wheel_part: system Motor_Gear_Wheel.impl;
            optical_encoder_part: system Optical_Encoder.impl;
            compass_imu_part: system COMPASS_IMU.impl;
            left_channel_1_part: system Left_Channel_1.impl;
            bustype_6_1: bus BusType_6.impl;
            bustype_9_1: bus BusType_9.impl;
            can_1: bus CAN.impl;
            bustype_8_1: bus BusType_8.impl;
            bustype_7_1: bus BusType_7.impl;
            bustype_3_1: bus BusType_3.impl;
            bustype_5_1: bus BusType_5.impl;
            bustype_4_1: bus BusType_4.impl;

        connections
            bustype_6_1_1_connection: bus access buck_analog_part.pp_1 -> bustype_6_1;
            bustype_6_1_2_connection: bus access h_bridge_2_part.pp_1 -> bustype_6_1;
            bustype_6_1_3_connection: bus access h_bridge_1_part.pp_1 -> bustype_6_1;
            bustype_9_1_4_connection: bus access power_unit_part.bustype_9_1 -> bustype_9_1;
            bustype_9_1_5_connection: bus access buck_analog_part.pp_2 -> bustype_9_1;
            can_1_6_connection: bus access left_channel_part.can_1 -> can_1;
            can_1_7_connection: bus access gps_sensor_part.can_1 -> can_1;
            can_1_8_connection: bus access power_unit_part.can_1 -> can_1;
            bustype_8_1_9_connection: bus access h_bridge_1_part.pp_3 -> bustype_8_1;
            bustype_8_1_10_connection: bus access power_unit_part.bustype_8_1 -> bustype_8_1;
            ethernet_1_11_connection: bus access left_channel_part.ethernet_1 -> ethernet_1;
            ethernet_1_12_connection: bus access camera_part.pp_1 -> ethernet_1;
            ethernet_1_13_connection: bus access ethernet_router_part.pp_1 -> ethernet_1;
            bustype_7_1_14_connection: bus access h_bridge_1_part.pp_2 -> bustype_7_1;
            bustype_7_1_15_connection: bus access motor_gear_wheel_part.pp_1 -> bustype_7_1;
            bustype_3_1_16_connection: bus access power_unit_part.bustype_3_1 -> bustype_3_1;
            bustype_3_1_17_connection: bus access gps_sensor_part.bustype_3_1 -> bustype_3_1;
            bustype_5_1_18_connection: bus access optical_encoder_part.pp_1 -> bustype_5_1;
            bustype_5_1_19_connection: bus access power_unit_part.bustype_5_1 -> bustype_5_1;
            bustype_4_1_20_connection: bus access compass_imu_part.pp_1 -> bustype_4_1;
            bustype_4_1_21_connection: bus access power_unit_part.bustype_4_1 -> bustype_4_1;
            vbat_hp_1_connection: feature power_unit_part.vbat_hp -> left_channel_part.vbat_hp;
            images_2_connection: feature camera_part.cp_1 -> left_channel_part.images;
            camera_control_3_connection: feature left_channel_part.camera_control -> camera_part.cp_2;
            location_4_connection: feature gps_sensor_part.location -> power_unit_part.location;
            v_5_connection: feature v -> left_channel_part.v;
            w_6_connection: feature w -> left_channel_part.w;
            wheels_control_7_connection: feature left_channel_part.wheels_control -> power_unit_part.wheels_control;

        properties
            Actual_Connection_Binding => (reference (vbat_hp_1_connection)) applies to can_1_part;
            Actual_Connection_Binding => (reference (images_2_connection)) applies to ethernet_1_part;
            Actual_Connection_Binding => (reference (camera_control_3_connection)) applies to ethernet_1_part;
            Actual_Connection_Binding => (reference (location_4_connection)) applies to bustype_3_1_part;
            Actual_Connection_Binding => (reference (wheels_control_7_connection)) applies to can_1_part;

    end TwIRTee.impl;

end TwIRTee;
```



```
package PA_TwIRTee

public
    with Twirtee;
    with PA_left_channel; renames PA_left_channel::all;
    with PA_GPS_Sensor; renames PA_GPS_Sensor::all;
    with PA_Power_Unit; renames PA_Power_Unit::all;
    with Base_Types; renames Base_Types::all;

    system twirtee extends Twirtee::twirtee
        features
            v: refined to in data port Integer;
            w: refined to in data port Integer;
    end twirtee;

    system implementation twirtee.pa extends Twirtee::twirtee.impl
        subcomponents
            left_channel_part: refined to system Left_Channel.pa;
            gps_sensor_part: refined to system GPS_Sensor.pa;
            power_unit_part: refined to system Power_Unit.pa;
    end twirtee.pa;

end PA_TwIRTee;
```

Software Architecture with AADL


AADLInspector (Z:/pierre.gaufillet/git/twirtee/design/twirtee-physical-architecture-aadl/packages/twirtee_data_types.aadl)
File View Wizards Tools ?
physical_architecture
power_manager
power_unit
twirtee
twirtee_data_types
Static Analysis
Schedulability
LMP
Ocarina
AI Scripts

```
package TwIRTee_Data_Types
public

        with Data_Model;

        data Image
                properties
                        Data_Model::Data_Representation =
        end Image;

        data Command
                properties
                        Data_Model::Data_Representation =
        end Command;

        data Camera_Command extends Command
        end Camera_Command;

        data Cartography
                properties
                        Data_Model::Data_Representation =
        end Cartography;

        data Location
                properties
                        Data_Model::Data_Representation =
        end Location;

end TwIRTee_Data_Types;
```

```
----------------------------------------------------
aadlrev2.7 (c)Ellidiss Technologies 21Feb2015
AADL-2.1
----------------------------------------------------
            *** INSTANCE MODEL ***
----------------------------------------------------
Root System Instance: Physical_Architecture::Physical_System.pa
----------------------------------------------------
  433 (system)..... root
----------------------------------------------------
Number of Component Instances:
----------------------------------------------------
- Processors:  0
- Processes:   0
- Threads:     0
- Subprograms: 0
----------------------------------------------------
           *** DECLARATIVE MODEL ***
----------------------------------------------------
Number of Packages:...........................18
Number of Component Types:....................35
Number of Component Implementations:..........24
Number of Subcomponents:......................21
Number of Call Sequences:......................7
Number of Subprogram Calls:....................7
Number of Features:...........................40
Number of Connections:........................20
Number of Property Associations:..............45
---- Prototypes ------------------------------------
Number of Prototypes:..........................0
Number of Prototype Bindings:..................0
```

- Validation with AADL inspector
  - Structural properties
  - *Scheduling with Marzhin and Chedar (TBD)*

Distributed applications Framework with PolyORB

- OCARINA generates code skeletons for each application from the software architecture in AADL including:
  - Tasks management
  - Synchronization/Communication glue
- The generated code relies on a distributed application framework: PolyORB HI (C, ADA)
- The functional code is completed by hand

Capella2AADL in brief

- Role
  - Initializes hardware/software design from system physical architecture
- Job done
  - AADL analysis and meta-modeling
  - AADL Prototyping
  - Experimentation on TwIRTee
- Issues
  - Multiplexed communications needed in AADL not supported by Capella
- Limitations
  - Global transformation only – Subsets not supported
- Capella to AADL Roadmap
  - Support transparent components on physical flows – transceivers, routers, etc.
  - Improved naming – ports, flows, etc.
- Open the door to better exchanges of information between system and software engineers

Questions ???

Thanks a lot for your attention