Model Driven Engineering with Capella and AADL

Bassem Ouni[1], Pierre Gaufillet[1, a], Eric Jenn[1, b], Jérôme Hugues[2]

1: IRT Saint-Exupéry, 118 route de Narbonne, 31042 Toulouse, France

2: ISAE, 10 Avenue Edouard Belin – BP 54032 - 31055 TOULOUSE CEDEX 04

Abstract— **The development of real time embedded equipments is a challenging task that requires the elaboration of multiple models in several domains, notably system, electronics and software, spanning a large spectrum of multiple abstraction levels and viewpoints: structural, behavioral, dependability, etc. These models serve various purposes: specification, design, evaluation or verification and validation. Today, no single modeling language and environment covers all these aspects. While Capella – an open source modeling language and environment for system engineering developed by Thales – fits well to the most early stages of the development process, AADL – the *Architecture Analysis and Design Language* defined by the *Society of Automotive Engineers* – provides powerful capabilities to describe and analyze the design artifacts of the software point of view that appear during the latest phase of the design. This is why they have both been selected in the project INGEQUIP of IRT Saint Exupéry. While using different modeling languages for different purpose is perfectly acceptable in a development process, it is important to guarantee that information remain consistent across all models. This is why building a formalized bridge between Capella and AADL is an essential piece of INGEQUIP process. In this paper, after an introduction to the context of INGEQUIP, the high level semantics of Capella and AADL are compared. The mapping used in INGEQUIP between Capella physical models and AADL abstract models is then described. The whole approach is illustrated by some elements coming from the design of TwIRTee – the robotic demonstrator of INGEQUIP – before concluding.**

Keywords—Real time, Embedded systems, Model transformations, Performance evaluation, Constraints verification, Capella, AADL.

I. INTRODUCTION

This work is achieved under the INGEQUIP project at the Toulouse *Institut de Recherche Technologique* (IRT) Saint-Exupéry. IRTs are new research structures established under the auspices of the French *Agence Nationale de la Recherche* (ANR). IRTs are aimed at favouring the transfer of innovation from laboratories to industries. Towards this goal, IRTs gather engineers coming from small to large companies from various industrial domains, and researchers from public universities and national research agencies. As an example, INGEQUIP covers the space, aeronautics and automotive systems domains.

The goal of INGEQUIP is to study and propose solutions for supporting closely integrated development of the main technical domains involved in embedded equipment engineering – system, eleectronics and software engineering. A key element for reaching such goal is to ensure the continuity and consistency of information in the whole chain of activities. In INGEQUIP, the choice has been to obtain this property by relying on models and model transformations. The set of requirements regarding an equipment is usually divided into two categories: functional requirements and non functional requirements. Functional requirements include the system's behavior, capabilities and characteristics as specified by stakeholders whereas non-functional properties or requirements define criteria that can be used to evaluate the operations of the system. In order to design the system and associate to the design elements the realized functional and non functional requirements, several modeling languages are available among which AADL

a Seconded from Airbus, 316 route de Bayonne, 31000 Toulouse, France.
b Seconded from Thales Avionics, 105 avenue du Général Eisenhower, 31100 Toulouse, France.

[1] – *Architecture and Analysis Design Language*, AUTOSAR [2], Capella [3], EAST-ADL [4], SysM [5] and UML [6] have been considered. While Capella, EAST-ADL and SysML fit system engineering, AADL, AUTOSAR and UML are focused on software engineering. Finally the couple Capella/AADL has been chosen in INGEQUIP because they provide most viewpoints commonly used by designers of embedded equipment at system and software levels: functional breakdown, logical and physical architecture, software dynamic and static architecture, formal behavioral descriptions; they are supported by a number of tools widely available: Capella and OSATE for edition, formal behavioral analysis tools like FIACRE and Tina [7], Cheddar [8] and code generations like OCARINA [9]). They are also based on open technologies like the Eclipse Modeling Framework which allows to develop easily tools extensions.
Consistently with the orientation of INGEQUIP, a transformation has then been defined and developed for ensuring a seamless transition from system engineering stage to software engineering stage.

In this paper, we therefore begin in section 2 by introducing the state of the art of engineering models transformation. Then, section 3 presents a first comparison between the semantics of Capella and AADL. The mapping of the transformation between Capella physical model and AADL abstract model is then described in section 4. The whole approach is illustrated by some elements coming from the design of TwIRTee – the robotic demonstrator of INGEQUIP – in section 5 before concluding in section 6.

II. RELATED WORKS

In order to achieve an early analysis of the specification, the verification of functional and non-functional properties of the system, and even code generation for the targeted hardware platform, several studies have proposed comparable transformations to AADL models.
In [10], Brun et al. introduce an approach for translating UML/MARTE detailed design into AADL design. The proposed work focuses on the transformation of the thread execution and communication semantics and does not cover transformation of embedded system components, such as equipment parts.
Turki et al. [11] propose a methodology for mapping MARTE model elements to AADL components. They focus on the issues related to modelling architectures. This transformation flow does not consider issues related to the mapping of MARTE properties to AADL property. The syntactic differences between MARTE and AADL are well handled by the transformation rules provided by ATL tool.
In [12], AADL is used for modelling the properties of embedded system architecture, including the application's tasks, the hardware platform and the operating system services in order to characterize the energy overhead of embedded operating system . The authors propose an AADL model transformation in order to be exploited by a multiprocessor simulation tool named STORM (Simulation TOol for Real-time Multiprocessor scheduling). AADL provides hardware and software architectures together with the scheduling policy; STORM simulates the system behaviour using all the characteristics (task execution time, processor functioning conditions, etc.) in order to obtain the chronological track of all the scheduling events that occurred at run time, and compute various real-time metrics in order to analyse the system behaviour and performances from various point of views. Then, this work has been extended in [13] and a system design exploration methodology has been proposed to verify system requirements when allocating applicative tasks to the processors using a set of tools: RDALTE for the definition and analysis of system requirements and QAML for quantitative analysis.
In this paper, the proposed approach for transformation of Capella to AADL models target to cover the various levels of abstraction when modeling systems. We take into account the system behavior and the hardware/software mapping. Next section will detail the transformation flow and how it exploits the complementarity of Capella and AADL in order to cover various embedded system aspects at high level modeling step.

III. PROPOSED METHOD

As Capella and AADL partially overlap, as shown in the diagram (Figure 1) below, the first question that has to be answered before defining the transformations from Capella to AADL is the level at which this transformation should be performed. Capella is clearly positioned on the most abstract part of the system development process with the *Operational Analysis*, focusing on the capture of stakeholders' needs, and the *System analysis*, focusing on the functional definition of the system. As for AADL, it doesn't offer support for such kind of analysis. As they deal with the system's architecture, Capella's logical and physical models share lots of concepts with AADL, in particular the capability to express structural refinement. AADL however goes further by providing not only system-oriented abstract components, but also explicit hardware components – *device*, *processor*, *memory*, *bus* – and explicit software components – *process*, *thread*, *subprogram*.

Moreover, AADL proposes a formal models of communication and execution semantics that is a prerequisite for any low-level behavioral analysis (e.g., schedulability analysis).

Capella provides system modeling capabilities at several layers of abstraction:

- At operational level, the customer needs, the actors, the missions and the activities are described.
- At system level, a Capella model defines how system can satisfy the former operational need.
- Capella logical level modeling starts from functional and non-functional analysis and builds one or several decompositions of the system into logical components.
- The building of logical components is performed at physical level: the "final" architecture of the system introducing architectural patterns, services and components, and it makes the logical architecture evolve according to implementation, constraints and choices.

Figure 2 summarizes the relationship between functional, logical and physical architecture in a Capella model. In this work, we interest to parse the physical architecture of the Capella model as it includes functional, logical and physical components.

Whatever the abstraction level of the considered components is, AADL also brings the capability to specify additional information compared to Capella, like functional or non-functional properties. Consequently, the simplest articulation between the two languages is at the levels of the logical or physical architectures. Considering that Capella is well adapted to describing the logical architecture and a first abstract level of physical architecture, and that it is a good practice to limit the risk of data duplication and inconsistency between Capella and AADL models, the best solution is to delay as far as possible the transfer of information. As shown in figure 3, the model transformation will therefore take as input the most detailed physical architecture in Capella, and the design process will go on in AADL.

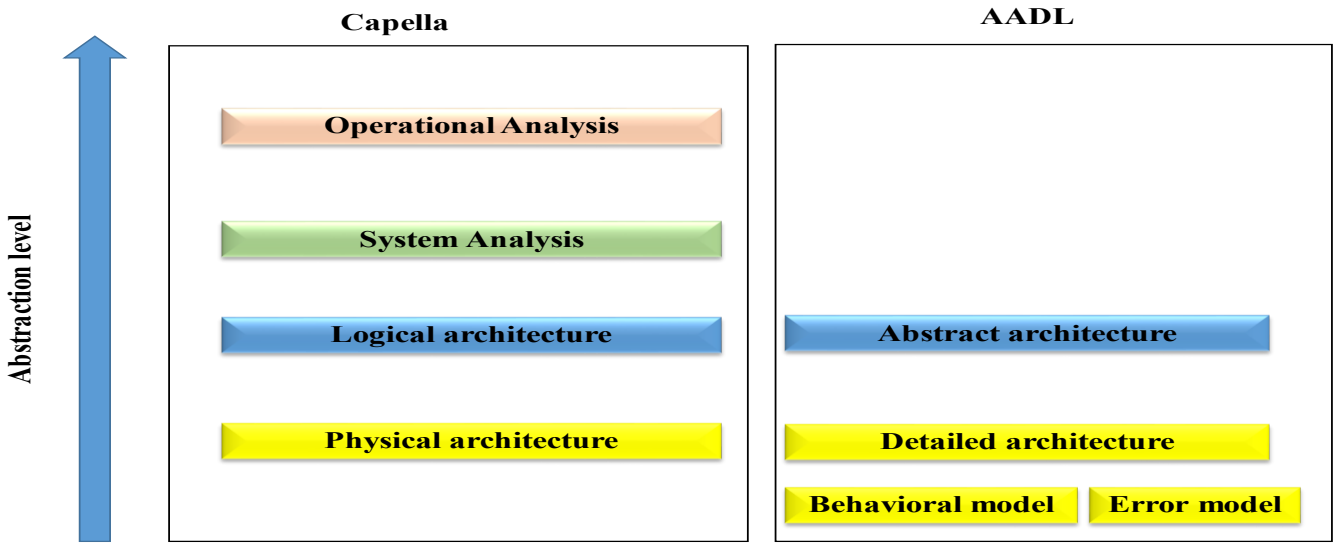

Figure 1- Capella and AADL positioning

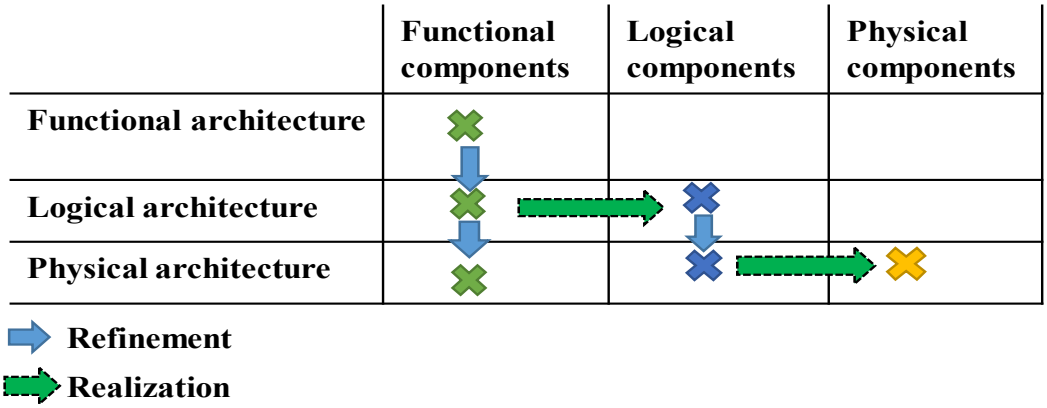

Figure 2- Capella functional, logical and physical architectures

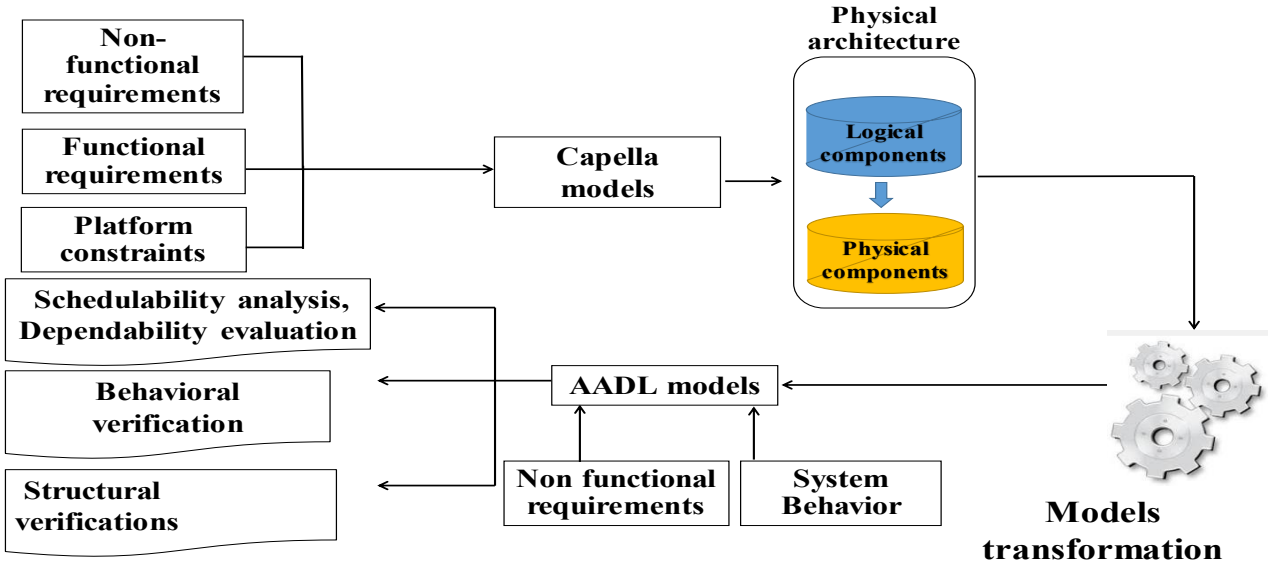

Figure 3 - Capella/AADL data flow

IV. THE CAPELLA TO AADL TRANSFORMATION

A. Transformation definition

To transform Capella to AADL models, we explore the contents of a Capella model and generate the appropriate AADL code following the Capella/AADL concepts analogy proposed in Table 1. The equivalent of an AADL processor, device are Capella node physical components with respectively a software execution unit and hardware kinds. A physical component with behavior nature and software application kind is considered as process in AADL.

Physical buses components are not modeled in Capella meta-models. For this reason, using Capella ecore meta-models, we generate Capella EMF [14] code and exploit it using Java to extract physical buses from Capella model and map them to AADL buses. The determination of buses is elaborated by exploring the Capella physical components, the physical links which are the communication/transportation means linking node Physical components, and physical ports. As depicted in figure 4, the physical links connected by physical ports are gathered in "*AllconnectedLinks"* list. For each element of the list, a physical bus component, with a bus port, will be generated to bind the physical links. Then, as showed in Figure 5, buses are connected to external ports (*tip ports*) which are ports of physical components having no subcomponents. This established link is called *BusLink*.

*BusLink*s are divided into segments that don't cross physical component boundaries. These segments represent synthetic links. Synthetic ports includes the *tipports* and new ports which are generated at each intersection between *BusLink*s and physical components. The algorithm describing buses extraction is detailed in algorithm 1. The output of this algorithm is the list of synthetic ports and links that will be mapped to AADL elements.

In Capella, logical components are not stored in the physical components they are deployed on, but, in the physical system root, , For rebuilding the AADL connection path, and as depicted in figure, a virtual physical component is created instead of each logical component, a connection is established between these virtual components, then as for the buses, we split this connection into parts and we generate new Capella ports with a new property: the direction.
To transform Capella to AADL models, Accelo plugin [15] is used, it allows the generation of AADL code from input Capella graphic models. We exploit this plugin to explore the contents of a Capella model and generate the appropriate AADL code following the Capella/AADL concepts analogy proposed in Table 1.

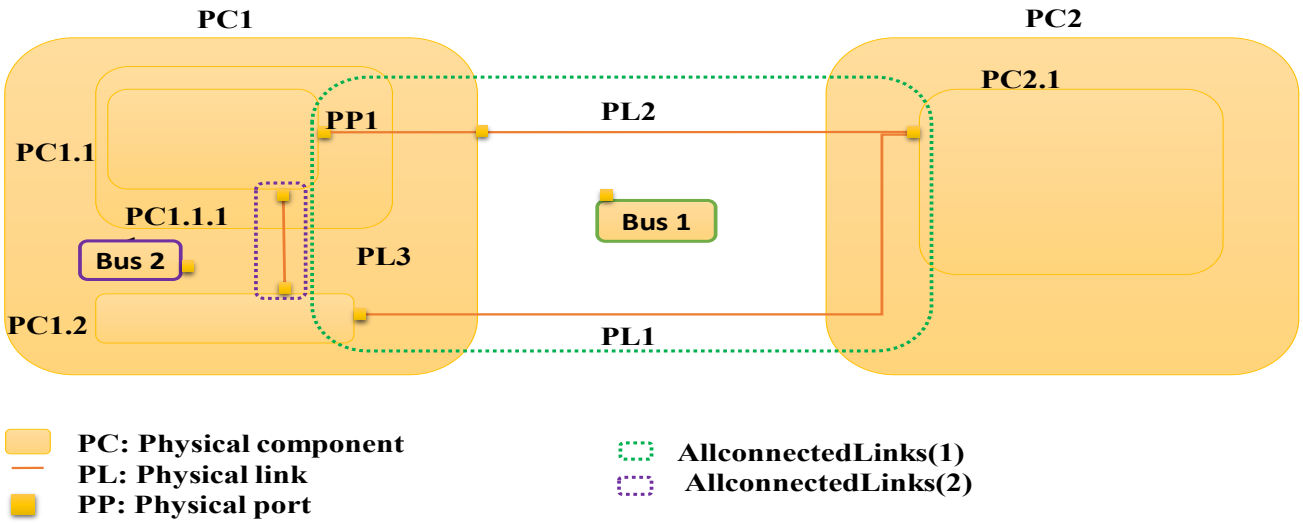

Figure 4 – Determination of connected physical links

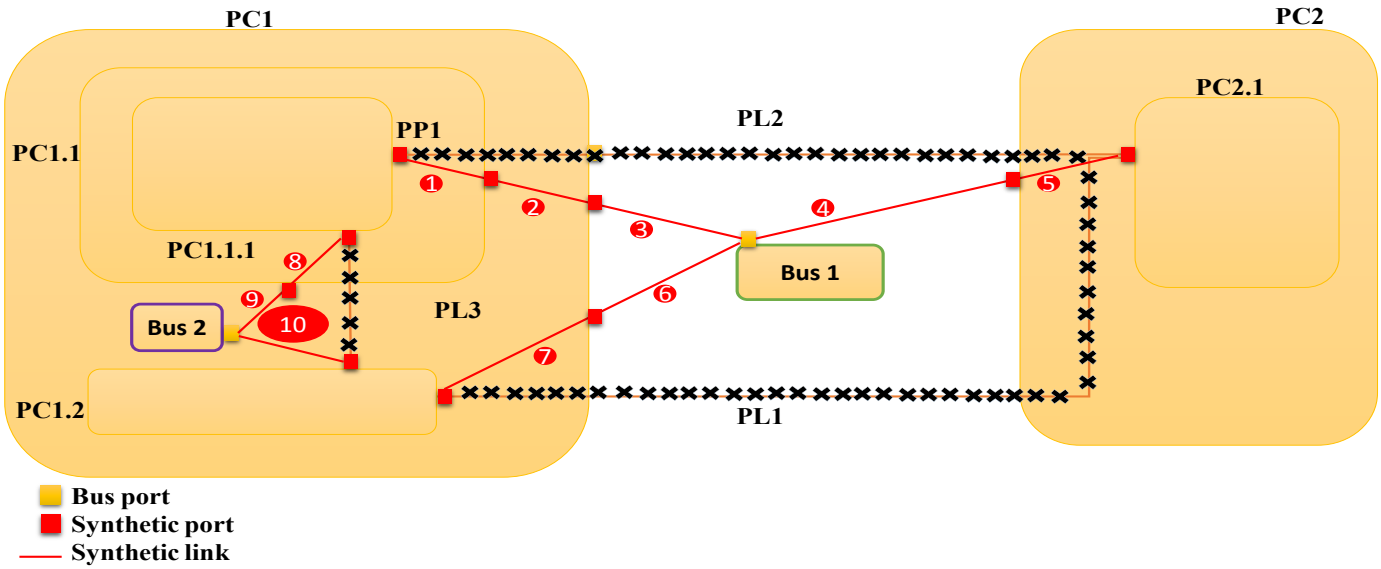

Figure 5 – Generation of synthetic links and ports

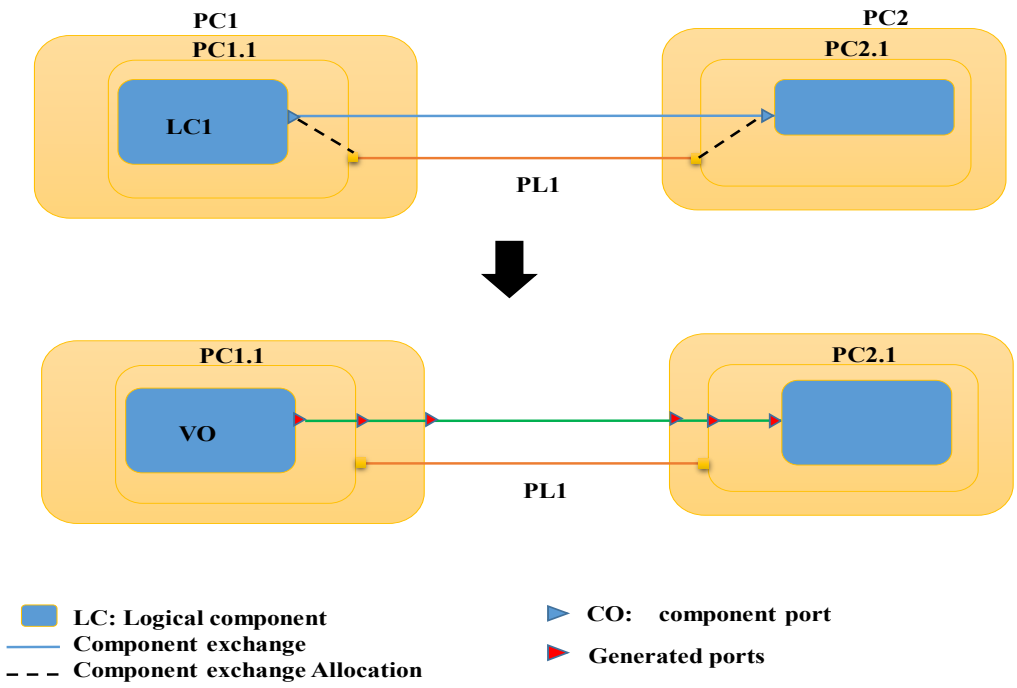

Figure 6 – Capella component exchanges transformation

Algorithm 1 – Capella Bus generation algorithm

---

1. Get the physical architecture from Project in Capella model.
2. Extract the list of physical ports ***pps*** physical links ***pls*** and a cross reference between them.
3. Determinate the list ***allConnectedLinks*** including the sets of ***pls*** connected by ***pps.***
4. **for** each element of ***allConnectedLinks***
    a. Get the category of the link
    b. Create the list of external ports ***tipports***
    c. Create a bus instance having the same name of category
    d. Associate the bus instances with ***tipports*** in a BiMap structure ***bus2TipPorts*** (key=bus instance, value=tipports).
5. **end for**
6. Initialize the list of synthetic ports with the tip ports and synthetic links.
7. **for each** element of ***bus2TipPorts***
    a. Search the closest common physical component ancestor ***PcCommonAncestor*** of ***tipPorts***
    b. Create the Physical component that will model physically the bus instance: ***PCBus***
    c. Add new properties to ***PCBus***: ***isBus*** and ***Bustype*** property
    d. Add a port **BusPort** to the ***PCBus***
    e. Add **PCbus** to **PcCommonAncestor**
    f. **for each** port ***p*** of ***bus2TipPorts***
        i. Create a link from tip port ***p*** to ***BusPort***
        ii. Divide this link into segments that don't cross Physical Component boundaries
        iii. Update the list of synthetic ports and links
    g. **end for**
8. **end for**

Case	Capella	Condition	AADL
A	PhysicalComponent	nature =PhysicaComponentNature::NODE and kind=PhysicalComponentKind::SOFTWARE_EXECUTION_UNIT	Processor
B	PhysicalComponent	nature =PhysicaComponentNature::NODE and kind=PhysicalComponentKind::HARDWARE	Device
C	PhysicalComponent	nature =PhysicaComponentNature::BEHAVIOR and kind=PhysicalComponentKind::SOFTWARE_APPLICATION	Process
D	PhysicalComponent	other than	System
E	PhysicalPort	See Bus extraction algorithm	Requires bus access
F	PhysicalLink		Bus access connection + Bus
G	ComponentPort	kind=ComponentPortKind::FLOW and direction=OrientationPortKind::[IN|OUT|INOUT] See Feature connection extraction algorithm	Feature
H	ComponentExchange	See Feature connection extraction algorithm	Feature connection
I	ComponentPortAllocation		Actual_connection binding_property on bus
J	Other metaclasses		N/A

Table 1 – Mapping of Capella – AADL concepts

V. APPLICATION TO THE TWIRTEE ROVER

TwIRTee is a three-wheeled autonomous rover developed within the INGEQUIP project. Its operational role is very simple: move itself on some predefined tracks from a point A to a point B (a "mission") while avoiding other rovers. To achieve this mission, it is fitted with several sensors (camera, odometry sensors, global positioning,…) and two main actuators (motors).

The development of the rover is not an objective *per se.* Indeed, TwIRTee is designed so as to cover the major topics addressed in the project namely: early validation, architecture exploration, performance prediction, and formal verification. Furthermore, it is aimed at covering issues, and functional and architectural elements specific to the three industrial domains. Accordingly, missions, functions and the architectural elements are determined so as to tackle or exercise one or several issues: for instance, “the localization” function relies partially on imaging so as to exercise hardware / software space exploration and co-design; the highly redundant architecture provides the experimental setup to perform early performance evaluations (including dependability), etc.

The computing platform of TwIRTee is composed of 2 COM/MON channels that host the main “mission” functions and one channel dedicated to power supply generation and motor control. In order to cover issues related to the development of safety-critical Man Machine Interfaces, the system also contains a remote operator station. Figure 3 and Figure 4 show respectively elements of TwIRTee design in Capella and their corresponding elements in AADL.

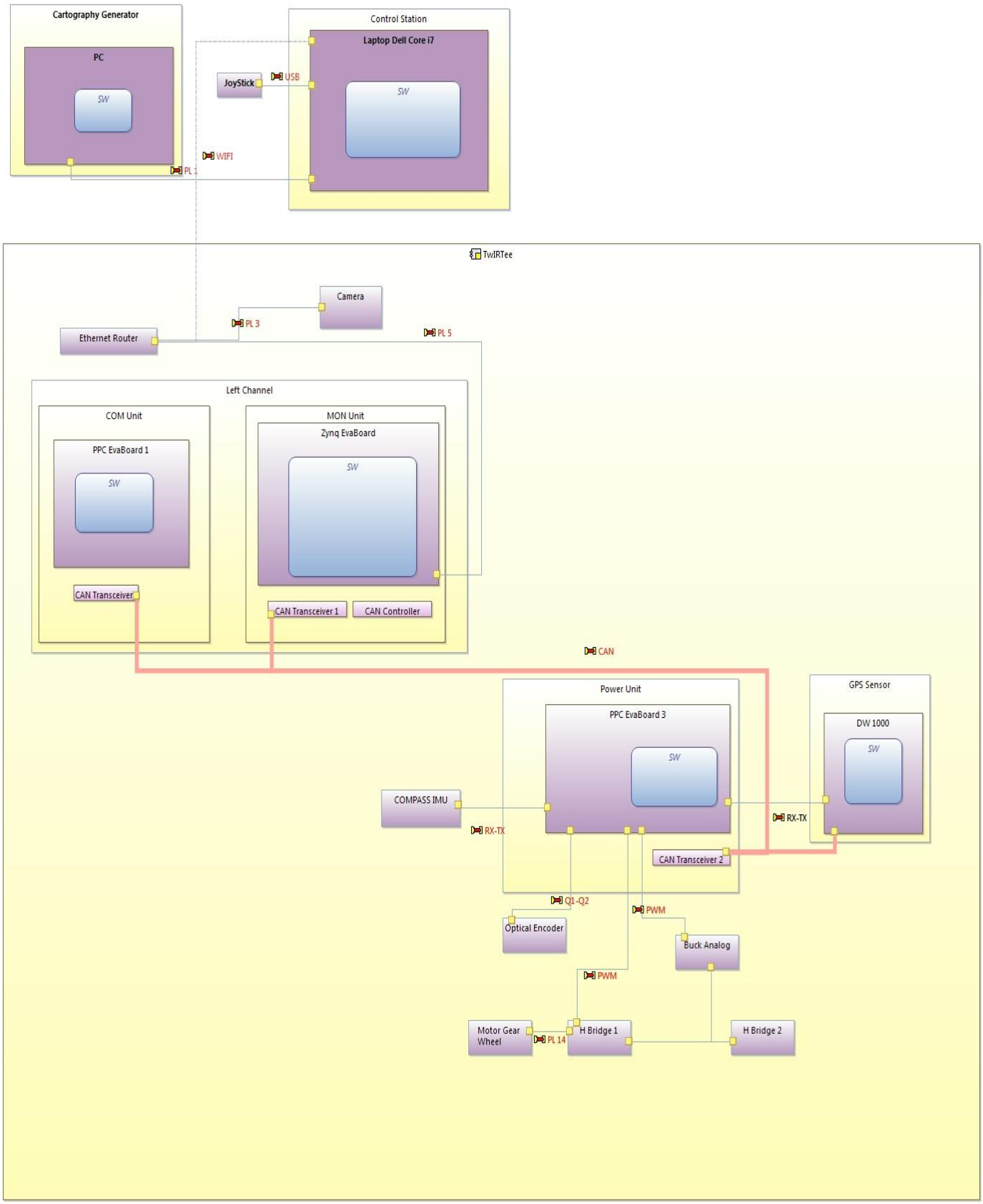

Figure 7 - Elements of TwIRTee design in Capella

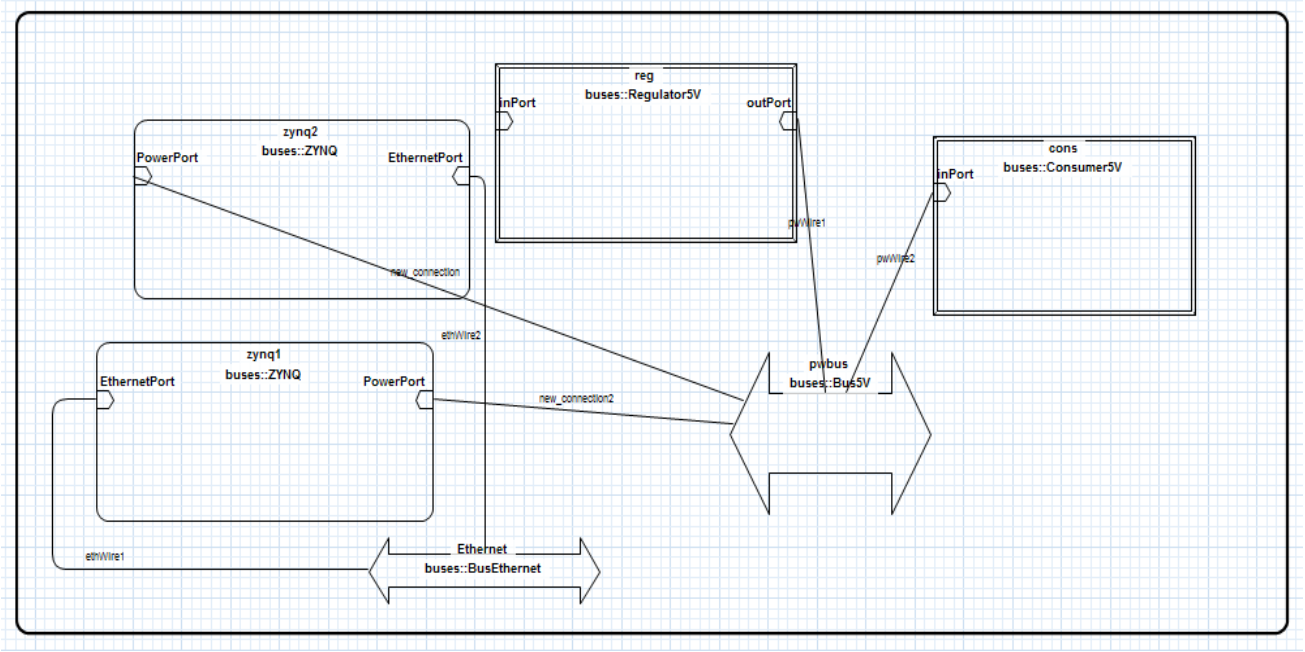

Figure 8 - Corresponding TwIRTee elements in AADL

VI. CONCLUSION

In this paper, we introduced a transformation from Capella physical architecture to a preliminary software architecture in AADL. The goal of this approach is to implement a seamless development process from system definition and design to software design and implementation. The approach has been applied and validated during the design of the TwIRTee rover demonstrator. The resulting physical model has been used to verify structural, performance (power and energy consumption), timeliness and dependability properties.

ACKNOWLEDGEMENT

The authors thank all people and industrial partners involved in the Ingequip project. This work is supported by the French Research Agency (ANR) and by the industrial partners of IRT Saint-Exupéry Scientific Cooperation Foundation (FCS): Actia Automotive, Airbus (a), Airbus Defense and Space, Continental Automotive, SAGEM, Systerel, Thales Avionics (b), ASTC Design Partners, Space Codesign Systems.

VII. REFERENCES

[1] Architecture Analysis and Design Langage (AADL), SAE standards, "Architecture Analysis and Design Langage (AADL), SAE standards," [Online]. Available: http://standards.sae.org/as5506/.

[2] "AUTOSAR," [Online]. Available: http://www.autosar.org/.

[3] "Capella tool environement and Arcadia methodology," Thales group, 2015. [Online]. Available: https://www.polarsys.org/capella/arcadia.html.

[4] "EAST-ADL," [Online]. Available: http://www.east-adl.info/Specification.html.

[5] "SysML," [Online]. Available: http://www.omgsysml.org/.

[6] "UML," [Online]. Available: http://www.uml.org/.

[7] B. Berthomieu, J.-P. Bodeveix, S. Dal Zilio, P. Dissaux, M. Filali, P. Gaufillet, S. Heim and F. Vernadat, "Formal Verification of AADL models with Fiacre and Tina," *Embedded Real-Time Software and Systems,* 2010.

[8] "Cheddar," [Online]. Available: http://beru.univ-brest.fr/~singhoff/cheddar/.

[9] Ocarina, "Ocarina," [Online]. Available: http://www.openaadl.org/ocarina.html.

[10] M. Brun, M. Faugère, J. Delatour and T. Vergnaud, "From UML to AADL: an Explicit Execution Semantics Modelling with MARTE," in *EMBEDDED REAL TIME SOFTWARE AND SYSTEMS*, Toulouse, January 2008.

[11] S. TURKI, E. SENN and D. BLOUIN, "Mapping the MARTE UML Profile to AADL," in *Models ACES-MB Workshop Proceedings*, Oslo, Norway, October 2010.

[12] B. Ouni, C. Belleudy and E. Senn, "Accurate energy characterization of OS services in embedded systems," *EURASIP Journal on Embedded Systems,* vol. 6, 2012.

[13] B. Ouni, "Thesis dissertation: High-level energy characterization, modeling and estimation for OS-based platforms," Sophia Antipolis, France, July 2013.

[14] "Eclipse Modeling Framework," [Online]. Available: https://eclipse.org/modeling/emf/.

[15] "Acceleo Project," Obeo company, 2006. [Online]. Available: http://www.eclipse.org/acceleo/.