A Comparative Analysis Between SysML and AADL When Modeling a Real-Time System

Quelita A. D. S. Ribeiro[1], Achim Rettberg[2](✉), Fabíola Gonçalves C. Ribeiro[3], and Michel S. Soares[1]

[1] Federal University of Sergipe Sao Cristovão, Sao Cristovão, Sergipe, Brazil
[2] University of Applied Science Hamm/Lippstadt and CvO University Oldenburg, Hamm, Germany
achim.rettberg@iess.org
[3] Federal Institute Goiano, Goiânia, Brazil

Abstract. System Architecture has a primary role in communication between stakeholders, in addition to planning and structuring the whole architectural process. Architecture Description Languages (ADLs) should be helping within architectural activities. However, most ADLs have not yet been widely used in industry. Another limiting factor for the effective use of ADLs is the difficulty of these languages in concretely expressing complex systems architecture. Considering this situation for the effective use of ADLs, UML has been often used in past years for architecture modeling. However, UML itself presents difficulties in representing characteristics which are pertinent to real-time systems, such as security or real-time restrictions. One of the advantages of UML is its extensibility, ability which allows creation of profiles. Thus, this work presents the Systems Modeling Language (SysML), a UML profile used for system architecture modeling. SysML and Architecture Analysis & Design Language (AADL) languages were both applied to a case and compared. As a conclusion, it was noticed that SysML is better than AADL when modeling abstract characteristics, such as decision making and loops functionality, which are not well-described in AADL.

Keywords: SysML · AADL · Real-time system · System architecture

1 Introduction

As complexity of real-time systems and embedded applications increase, there is the continuous need of more abstract representation of such systems. Modeling systems architecture is a challenging task, since these systems are not only of great magnitude, but are also significantly different [5]. Moreover, challenges and difficulties happen when developing these systems, due to specific characteristics, including mobility and security or real-time restrictions [6,7]. One of the main challenges to be faced in the development of solutions is to connect domain

© IFIP International Federation for Information Processing 2023
Published by Springer Nature Switzerland AG 2023
M. A. Wehrmeister et al. (Eds.): IESS 2019, IFIP AICT 576, pp. 27–38, 2023.
https://doi.org/10.1007/978-3-031-26500-6_3

specific requirements, expressed by business analysts, with specific technological solutions designed by software architects [1,3].

Since UML is a popular language in the software industry, and UML-based modeling tools are fully available [12], the UML community has been working with the purpose of presenting a way of modeling real-time system's properties. One of these efforts related to embedded systems is the SysML (Systems Modeling Language) profile. SysML offers additional resources to UML, including requirements modeling [7,9,11], and specification of several structural, behavioral and temporal aspects of real-time systems [5,10].

Characteristics of SysML have been recognized for requirements engineering [2,7] and for real-time systems requirements modeling [5]. These advantages were evidenced and have been analyzed for modeling systems' architecture [4, 12]. In this paper we present a case on an automatic headlight modeling in an automotive system in two languages, SysML and ADDL. The objectives of developing architectural representation with two different languages are: (1) to analyze the weak and strong points of each language, (2) to offer a comparative example and (3) to propose SysML as an architectural description language.

Authors of paper [12] describe a survey among 60 possible ADLs for development of automotive systems. After analysis, 3 languages for architectural description were chosen: AADL, MARTE and SysML. MARTE (UML profile) was used in a complementary way to SysML in order to model the systems execution time. However, the author considers AADL as a better alternative for architectural representation, based in criteria of code generation, formal verification, error modeling and variability modeling.

A survey of modeling languages of real-time software systems is presented in [4]. For the authors, each language has its advantages and disadvantages and, in order to properly describe the architecture of the real-time software system, is almost certain the need of complementary use of two or more languages. In the referred work, AADL, UML, SysML and MARTE are explored in the context of comprehending the contribution that each language brings to software engineering and to determine if it is viable to combine aspects of the four modeling languages in order to obtain a wider coverage in architectural descriptions.

2 Headlight Control System Details

The software requirements presented in this Section were primarily collected through reading and research in automobile manuals, web-site and the book [13]. After this stage the requirements were identified and organized for the automatic headlight control system.

In the headlight control system, the headlights are activated through a photoelectric sensor. The sensor is activated or deactivated by lightning conditions, and may be dark or light. Thus, the automatic control of headlights switches on the lights whenever the sensor feels the dark environment. For instance, the system will activate the headlights when a car enters in a tunnel or when an environment presents heavy clouds.

In this example, according to the characteristics previously listed, and to the results of the detailed domain analysis, the functional and non-functional system requirements are described in Table 1:

Table 1. Description of the system requirements

ID	Description
FR1	The light control system shall turn off the front lights in exactly 3 min after engine is shut down
FR2	The light control system shall schedule simultaneous requests in memory every 50 ms
FR3	The light control system shall evaluate the functioning of all components every 50 ms
FR4	The light control system shall recognize sensor status of the front lights every 50 ms
FR5	The light control system shall recognize when the sensor detects a dark environment in a maximum of 50 ms
FR6	If the FR5 is satisfied, the light control system shall turn on the front lights in a maximum of 50 ms
FR7	The light control system shall recognize when the sensor detects light environment in a maximum of 50 ms
FR8	If the FR7 is satisfied, the light control system shall turn off the front lights in a maximum of 50 ms
FR9	The light control system shall get signal of the lights sensor every 50 ms
FR10	The light control system shall recognize the status from light actuator every 50 ms
FR11	The light control system shall notify to user the malfunction of any light component, with a light on the dashboard, in a maximum of 50 ms
FR12	The light control system shall modify light intensity[1] in a maximum of 50 ms, when there is variation of external lighting
FR13	The light control system shall allow the driver to turn off the automatic lights
FR14	The light control system shall allow the driver to turn on the automatic lights
FR15	The light control system shall allow the driver to turn on the lights manually
FR16	The light control system shall allow the driver to modify manually the lights intensity
FR17	The light control system shall allow the driver to turn off the lights manually
FR18	When driver uses the lights manually, the light control system shall notify to user if the driver leaves the lights on after engine is switched off (by audio)
FR19	The light control system shall run requests of requirement FR2 every 50 ms
NFR01	The car shall have reliability lights system, tests shall guarantee lowest rate (0.10%) to software failures
NFR02	When the car is in operation, the light control system shall be available 99% of the time while the car is on

3 Headlight System Modeled with SysML

SysML Requirements diagram allows each requirement to have an identification (ID) and a textual information using natural language in order to describe the requirement.

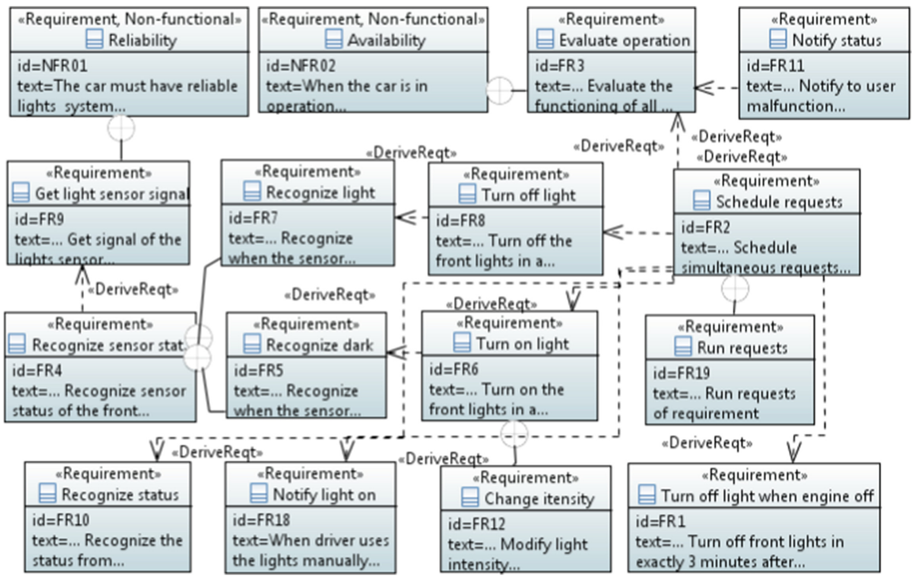

«Requirement, Non-functional» Reliability
id=NFR01
text=The car must have reliable lights system...
«Requirement, Non-functional» Availability
id=NFR02
text=When the car is in operation...
«Requirement» Evaluate operation
id=FR3
text=... Evaluate the functioning of all ...
«Requirement» Notify status
id=FR11
text=... Notify to user malfunction...
«Requirement» Get light sensor signal
id=FR9
text=... Get signal of the lights sensor...
«Requirement» Recognize light
id=FR7
text=... Recognize when the sensor...
«Requirement» Turn off light
id=FR8
text=... Turn off the front lights in a...
«Requirement» Schedule requests
id=FR2
text=... Schedule simultaneous requests...
«Requirement» Recognize sensor stat
id=FR4
text=... Recognize sensor status of the front...
«Requirement» Recognize dark
id=FR5
text=... Recognize when the sensor...
«Requirement» Turn on light
id=FR6
text=... Turn on the front lights in a...
«Requirement» Run requests
id=FR19
text=... Run requests of requirement
«Requirement» Recognize status
id=FR10
text=... Recognize the status from...
«Requirement» Notify light on
id=FR18
text=When driver uses the lights manually...
«Requirement» Change itensity
id=FR12
text=... Modify light intensity...
«Requirement» Turn off light when engine off
id=FR1
text=... Turn off front lights in exactly 3 minutes after...
«DeriveReqt»

Fig. 1. SysML requirements diagram of the headlight system.

The Headlight control Requirements diagram is depicted in Fig. 1. The diagram includes an incomplete description because, due to space limitation, the description was omitted. In Sect. 2 there is a complete textual description of the requirements. Functional requirements 13, 14, 15, 16, and 17 were not shown in Fig. 1 because these are manual requirements, and the concern in this paper is automatic requirements.

Relationship *"DeriveRqt"* relates a derivative requirement with its origin requirement. *"Containment"* relationship, (represented by the symbol ⊕—— in diagram in Fig. 1) specifies the hierarchy between requirements, its use prevent the reuse of requirements in different contexts, once a specific element of the model can only exist in a *"Containment"*. *"Trace"* relationship is of general purpose between a requirement and any other element of the model and its use occurs only for traceability reasons [8].

SysML also allows the representation of requirements, their properties and relationships in a tabular format which can be seen in Table 2.

Table 2. Requirements table (SysML) of the headlight system.

Id	Name	Type	Derived Req	DerivedFrom Req	Containment Req
FR8	Turn off light	Functional	FR2	FR7	
FR10	Recognize status	Functional	FR2		
FR11	Notify status	Functional		FR3	
NFR01	Reliability	Non-functional			FR9
NFR02	Availability	Non-functional			FR3
FR1	Turn off light when engine off	Functional	FR2		
FR2	Schedule requests	Functional		FR1, FR3, FR6, FR8, FR10, FR18	FR19
FR6	Turn on light	Functional	FR2	FR5	FR12
FR4	Recognize sensor status	Functional		FR9	FR5, FR7
FR5	Recognize dark	Functional	FR6		
FR3	Evaluate operation	Functional	FR2, FR11		
FR18	Notify light on	Functional	FR2		
FR7	Recognize light	Functional	FR8		
FR9	Get light sensor signal	Functional	FR4		
FR12	Change intensify	Functional			
FR19	Run requests	Functional			

3.1 Block Definition Model

SysML provides two diagrams which are useful for system architecture modeling: Block Definition Diagram (BDD) and Internal Block Diagram (IBD).

In the case study, BDD was developed for modeling components which are involved in the headlight system. The specified functions are implemented for secure communication with other components and with local sensors/actuators. SysML BDD is represented in Fig. 2.

A SysML BDD can describe general structural elements, varying from small to very large. SysML Blocks can be used to represent the hardware architecture, the data and procedures of a system [8]. In the example of the headlight system, 10 related blocks are defined. The relationship between blocks was represented by composition associations. The composition instance is synchronous, that is, if an instance is destroyed, finishing the execution, the other executions are also going to be finished. *"Clock"* Block controls time synchronicity. *"Light system"* block communicates with other blocks which are not directly part of the headlight system through the *"Communication control"* block, this communication is essential for the necessary information to the execution of the headlight system to be conducted.

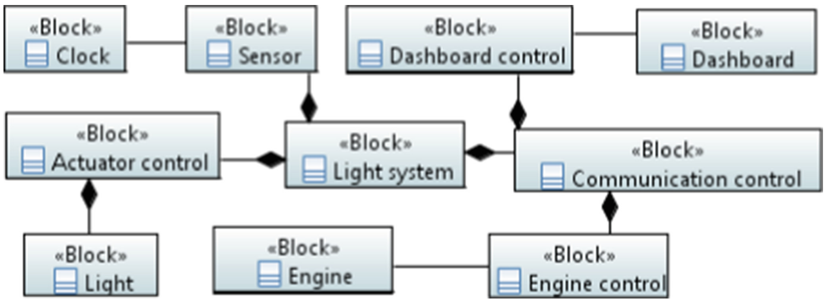


Fig. 2. BDD headlight system.

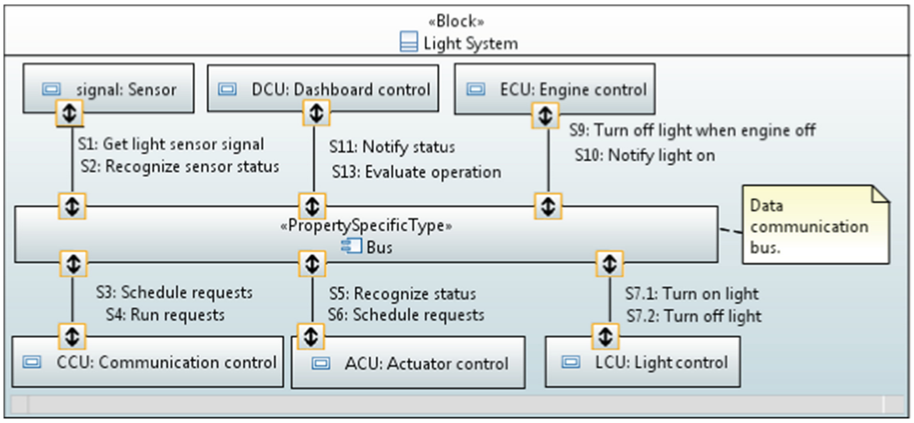


Fig. 3. IBD headlight system.

3.2 Internal Block Model

SysML IBD (Fig. 3) models the internal operations of the *"Light System"* block. The doors connect to external entities and interact with a block through connectors. A flow door specifies the entrance and exit items which can flow between a block and its environment. The doors specify the flow path, data communication, operations, receptions and hardware resources (BUS). For modeling a flow path, *"FlowPort"* was used, specifying entrance and exit items which can flow between a block and its environment. Flow doors relay entrance and exit items for a connector which connects blocks to the internal parts. The connectors are labeled by messages which represent operations between two parts.

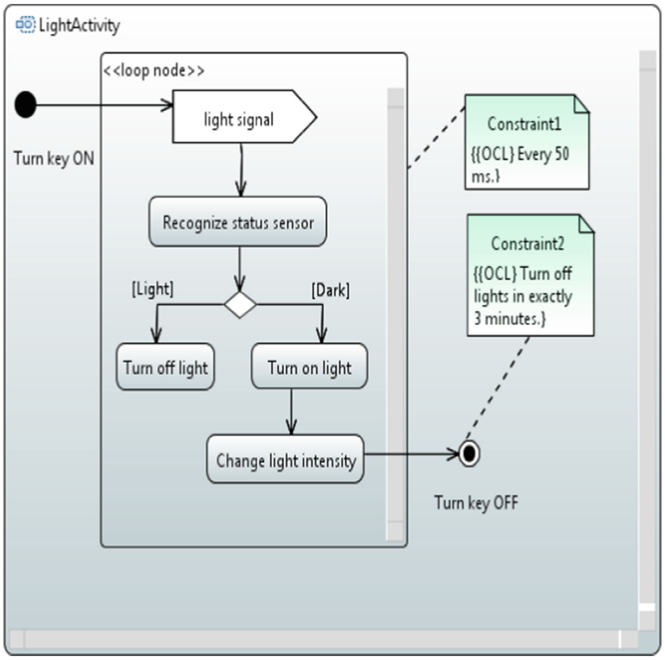


Fig. 4. Activity diagram of the headlight system.

3.3 Activity Model

The main headlight activity is depicted in Fig. 4. Initial and final activities were modeled, decision node which represents the action of switching the headlights on and off, interactive loop with ⟨⟨loop node⟩⟩ and restrictions with OCL for repetition of the loop and to switch the headlights off in exact 3 min after the car is switched off.

4 Headlight System Modeled with AADL - External Specification

AADL descriptions consist in basic elements named components. The modeling of the interface (External specification) of components is given by the AADL type definition. A component type declaration defines the interface elements of a component and externally observable attributes, such as devices which are part of the interaction with other components and connections (see Fig. 5).

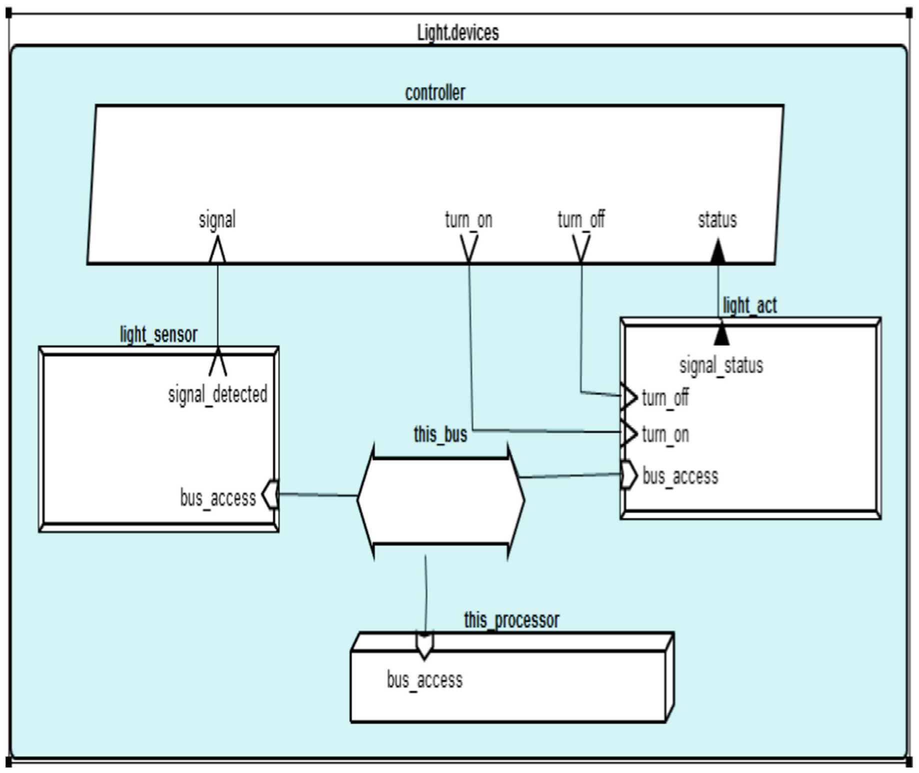
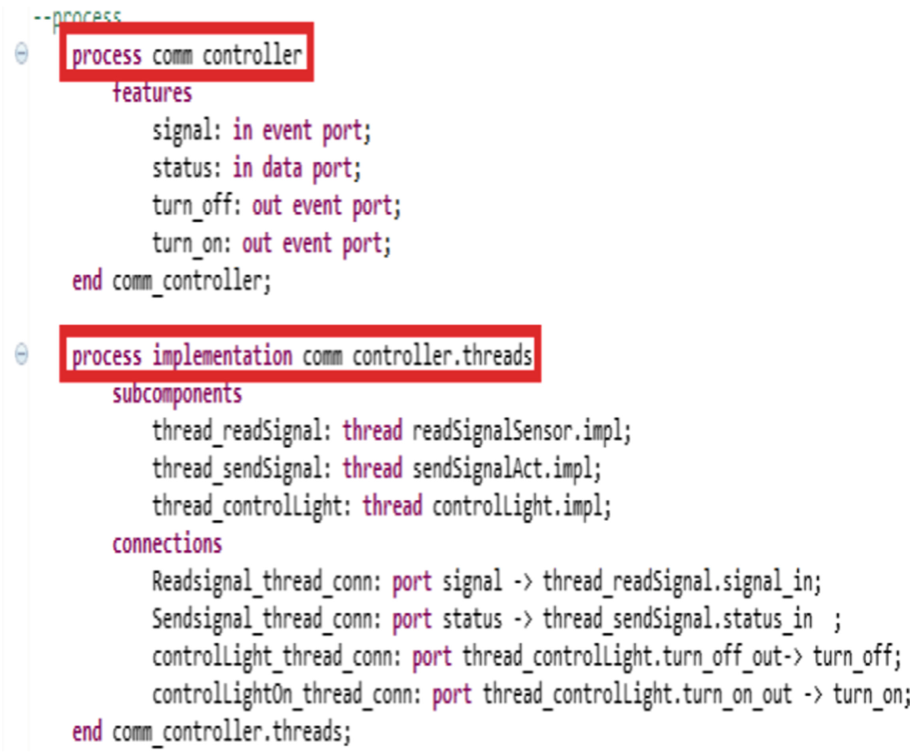


Fig. 5. External specification of the headlight control system in AADL.

The headlight control system, as shown in Fig. 5, is described in the AADL language. The system consists in a system unity, which contains a processor, a process and two devices. The system unity receives entrance signals coming from a sensor device named as *"light_sensor"*. When the absence or presence of light is detected, the *"light_sensor"* sends a signal through a door to the process named *"controller"*. The process receives a signal and the system starts to be executed with the information which was transmitted by the sensor. With the goal of executing a software, it was used a memory processor. The memory is a subcomponent of the processor, in memory the code is stored while the processor is in execution. The *"this_bus"* is required for loading data, controlling signals and establishing data and events interchange between hardware and software components, such as in the communication between the process and the sensor.



```
--process
process comm controller
    features
        signal: in event port;
        status: in data port;
        turn_off: out event port;
        turn_on: out event port;
end comm_controller;

process implementation comm controller.threads
    subcomponents
        thread_readSignal: thread readSignalSensor.impl;
        thread_sendSignal: thread sendSignalAct.impl;
        thread_controlLight: thread controlLight.impl;
    connections
        Readsignal_thread_conn: port signal -> thread_readSignal.signal_in;
        Sendsignal_thread_conn: port status -> thread_sendSignal.status_in  ;
        controlLight_thread_conn: port thread_controlLight.turn_off_out-> turn_off;
        controlLightOn_thread_conn: port thread_controlLight.turn_on_out -> turn_on;
end comm_controller.threads;
```

Fig. 6. Process code in AADL.

4.1 Internal Specification of AADL Headlight System

Internal specification is specialized by an external implementation. A components implementation declaration defines an internal structure of a component in terms of subcomponents, subcomponents connections, subprogram calls sequencies, modes, flow implementations and properties. Usually, external and internal specifications are performed by alternately repeating two stages. Source code of Fig. 6 presents an internal definition and the implementation of an AADL process. The process, named *"comm_controler"*, and the implementation of the process contains and controls the threads.

In the headlight control system, *threads* are subcomponents of the process, that is, they are an internal specification of the process and are presented in Fig. 7. These *threads* must be able to receive as entrance the light intensity value coming from the (*"thread_readSignal"*) sensor, to process the signal and transmit to the actuator the response regarding the automatic ignition of the headlights (*"thread_controlLight"*). These threads are connected with event doors, because this kind of door allows the rowing of the event associated data. The last thread must receive, as response from actuators, the current state of the headlights (*"thread_sendSignal"*).

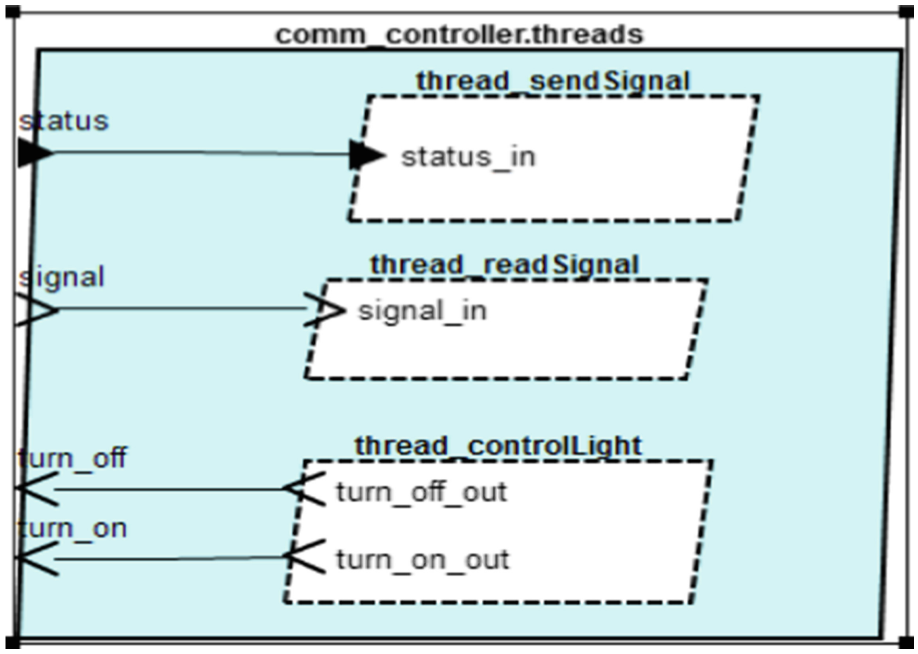


Fig. 7. Internal specification of the headlight control system in AADL.

A highlight of the AADL language is that, besides allowing both textual and graphical description of the system, it relates both.

5 A Comparative Analysis: SysML x AADL

In practical terms, it was observed that abstract features such as decision making, repetition of a functionality (loop), start and end representation of an activity, characteristics that are related to reality, and consequently, to the system, are not described in AADL. The lack of these characteristics was observed while the headlights were being modeled in AADL.

Currently, AADL does not provide a specific diagram/representation to show requirements or behavior by means of actions, such as the SysML Activity diagram. In addition, it is not possible to show the relationship of requirements with the systems design in AADL, and also relationships between system design and software design.

Connectors in AADL are not differentiated, when two types of systems are related, one can only relate them as "extension", in other words, one system only extends another. There is no other classification for connections like there is in SysML, for example, "Trace, Derive, ou Containment".

In AADL, component is identified only by name, there is no Identifier (ID). ID is essential to identify and track easily a requirement, as shown in the SysML example in Table 2.

Developing models in the AADL graphical language is a complex activity, it is necessary to manipulate the internal diagram so that the connections are changed in the external diagram, sometimes this manipulation of the internal diagram to the external or from the external to the internal becomes confused because of the ports and connections, connections are difficult to implement in the graphical model.

SysML provides diagrams for behavior modeling, such as the Sequence diagram, Activity diagram (Figure 4) and State Machine diagram. SysML obviously has strong ability to model system behavior. On the other hand, AADL provides a limited vocabulary for behavior modeling [12].

6 Conclusion

In this paper, the main focus is to show that the SysML language can be used to systems architecture modeling. The comparison between the SysML and AADL languages is performed to understand the limitations that SysML presents and to propose improvements in the architectural aspect.

It was observed that SysML can be used extensively for modeling abstract features. SysML accurately captures the requirements, management of system complexity is developed from the initial stages of production of a system with the expressiveness of the Requirements diagram.

Block diagram and Internal Block diagram capture the systems and subsystems, provide communication between them and the necessary resources. The Activity diagram can represent the steps required for a given activity to be completed successfully, or may display a predicted deviation. This diagram also captures the main strategic decisions of a system, and serves for communication between stakeholders.

We can conclude that the Requirements, Activities, Blocks and Internal Blocks diagrams are suitable for software architecture modeling. However, SysML needs to be expanded for classification of components, implementation of the model through textual syntax and description of time properties to have features necessary for modeling the architecture of real-time systems.

References

1. Van der Auweraer, H., Anthonis, J., De Bruyne, S., Leuridan, J.: Virtual engineering at work: the challenges for designing mechatronic products. Eng. Comput. **29**(3), 389–408 (2013). https://doi.org/10.1007/s00366-012-0286-6
2. Behjati, R., Yue, T., Nejati, S., Briand, L., Selic, B.: Extending SysML with AADL concepts for comprehensive system architecture modeling. In: France, R.B., Kuester, J.M., Bordbar, B., Paige, R.F. (eds.) ECMFA 2011. LNCS, vol. 6698, pp. 236–252. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21470-7_17
3. Brown, A.W.: Model driven architecture: principles and practice. Softw. Syst. Model **3**(4), 314–327 (2004). https://doi.org/10.1007/s10270-004-0061-2

4. Evensen, K., Weiss, K.: A comparison and evaluation of real-time software systems modeling languages. In: AIAA Infotech@ Aerospace 2010, p. 3504. American Institute of Aeronautics and Astronautics, California, USA (2010)
5. Khan, A.M., Mallet, F., Rashid, M.: Modeling systemverilog assertions using SysML and CCSL. In: Electronic System Level Synthesis Conference, ESLsyn Conference, Proceedings (2015)
6. Koopman, P.: Better Embedded System Software. Drumnadrochit Education, Pittsburgh (2010)
7. Marques, M.R.S., Siegert, E., Brisolara, L.: Integrating UML, MARTE and SysML to improve requirements specification and traceability in the embedded domain. In: Proceedings of the 12th IEEE International Conference on Industrial Informatics (INDIN), pp. 176–181. IEEE (2014)
8. OMG: OMG systems modeling language (OMG SysML). OMG Document: 03 June 2015, p. 346 (2015)
9. Ribeiro, F.G.C., Pereira, C.E., Rettberg, A., Soares, M.S.: Model-based requirements specification of real-time systems with UML, SysML and MARTE. Softw. Syst. Model. **17**(1), 343–361 (2016). https://doi.org/10.1007/s10270-016-0525-1
10. Ribeiro, Q.A.D.S., Ribeiro, F.G.C., Soares, M.S.: A technique to architect real-time embedded systems with SysML and UML through multiple views. In: 19th International Conference on Enterprise Information Systems (ICEIS), **2**(1), pp. 287–294 (2017)
11. dos Santos Soares, M., Vrancken, J.L.: Model-driven user requirements specification using SysML. JSW **3**(6), 57–68 (2008)
12. Shiraishi, S.: Qualitative comparison of ADL-based approaches to real-world automotive system development. Inf. Media Technol. **8**(1), 196–207 (2013)
13. Zurawski, R.: Embedded Systems Handbook, 2-Volume Set. CRC Press Inc, Taylor and Francis Group (2009)