A Technique to Architect Real-time Embedded Systems with SysML and UML through Multiple Views

Quelita A. D. S. Ribeiro[1], Fabíola G. C. Ribeiro[2] and Michel S. Soares[1]
[1]*Department of Computing, Federal University of Sergipe, Sao Cristovão, Sergipe, Brazil*
[2]*Federal Institute Goiano, Catalão, Brazil*

Keywords: Software Architecture, Automotive Embedded System, Real-time Systems, SysML, UML.

Abstract: Describing the architecture of real-time systems by means of semi-formal languages has been often considered in the literature. However, the most common approach is to propose multiple modeling languages in an orthogonal manner, i.e., the models are used in separate phases, in a totally independent way. This situation is not always possible, and the assumption in this paper is to propose a technique in which diagrams from two modeling languages are integrated. In this paper, UML and SysML are used together. Thus, the proposed technique is capable of modeling both software and system architectural elements, by satisfying the following modeling criteria: support to model components and connectors, both graphical and textual syntax, modeling non-functional requirements, design of structural view of software using UML classes, represent hardware elements in the architecture, and to describe traceability between requirements. A case study on a real-time automotive embedded system is presented to illustrate the technique.

1 INTRODUCTION

As complexity of electronic systems and embedded applications increase, there is a continuous need for more abstract representations of such systems. Modeling the architecture of systems is a challenging task as these systems are not only huge in magnitude but are also significantly diverse (Khan et al., 2015). Design of complex systems involves architectural specification of software, including software structure and organization, synchronization between tasks and processes, physical distribution of elements, composition of design elements, selection among design alternatives and considering quality characteristics such as scalability and performance. Software architecture is essential during all software development phases, given that the architecture description affects the success of a project and helps stakeholders to understand the software to be developed (Gardazi et al., 2009).

Embedded systems are dedicated computational systems, composed by hardware and software components. When the correctness of these systems depends not only on their functional behavior, but also on their timing behavior, they are classified as real-time embedded systems (Marques et al., 2014). UML, mainly through Class, Sequence, and Activity diagrams, is the most popular language for software modeling (Reggio et al., 2015). However, UML support for embedded systems presents many challenges, including the lack of standard formal semantics, many semantic variation points, and the variety of diagrams, which may contain inconsistencies. UML expressivity is considered not completely satisfactory for this domain (Jouault and Delatour, 2014).

These restrictions have motivated the investigation of strategies to successfully support the development of complex systems. Therefore, the Systems Modeling Language (SysML) offers additional resources to UML, including requirements and constraints modeling (Marques et al., 2014). SysML also has features to support the specification of diverse structural, behavioral and temporal aspects of complex embedded systems (Khan et al., 2015).

This work proposes a technique to use SysML as a modeling language in a real-time embedded system combined with UML. A qualitative analysis presents the advantages of this approach when comparing to other similar proposals published in the literature.

Ribeiro, Q., Ribeiro, F. and Soares, M.
A Technique to Architect Real-time Embedded Systems with SysML and UML through Multiple Views.
DOI: 10.5220/0006294802870294
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 2*, pages 287-294
ISBN: 978-989-758-248-6

2 BASICS ON SysML

SysML was developed by the Object Management Group (OMG) and International Council on Systems Engineering (INCOSE) with the objective of developing a unified general purpose language for modeling systems (SysML, 2015). SysML diagrams used in this paper are briefly as follows.

SysML Requirements diagram provides a modeling construct for text-based requirements (SysML, 2015). It models and describes several system requirements such as non-functional requirements, as well as showing the various types of relationships between different requirements, and other model elements that satisfy or verify them (SysML, 2015). SysML Requirements diagram can display requirements, packages, other classifiers, test cases, rationale and also relationships between requirements and also their relationships to other diagrams (SysML, 2015) (Soares and Vrancken,).

Table 1: SysML Airbag Requirements Table.

id	name	type	derived Req	derived-From Req	containment Req
FR8	Movement	functional	FR9	-	-
FR7	Angle	functional	FR9	-	-
FR9	Activate airbag	functional	-	NFR02, FR8, FR7	-
FR1	Get speed value	functional	RR2	-	-
FR4	Calculate weight	functional	FR3	-	-
FR2	Evaluate speed	functional	FR3, NFR06	FR1	-
FR3	Recognize slowdown	functional	FR5, FR4	FR2, NFR04	-
FR5	Calculate collision angle	functional	FR10	FR3	FR6
FR6	Recognize angle	functional	-	-	RF7, RF8
FR11	Get airbag sensor signal	functional	FR10	-	-
FR12	Recognize act status	functional	FR10	-	-
FR10	Schedule requests	functional	-	FR12, FR11, FR13, FR5	FR15
FR14	Notify status	functional	-	FR13	-
FR13	Evaluate operation	functional	FR10, FR14	-	-
NFR01	Safety	non-functional	-	-	FR1
NFR02	Reliability	non-functional	FR9	NFR05	-
NFR04	Availability	non-functional	FR3	-	-
NFR05	Rule	non-functional	NFR02	-	-
NFR06	Integrity	non-functional	-	FR2	-
FR15	Run requests	functional	-	-	-

SysML Blocks are modular units of system descriptions in SysML. SysML Blocks and their relationships are visualized by the SysML Block Definition diagram and their internal structure by the SysML Internal Block diagram. SysML Block Definition diagram (BDD from now on) is based on the UML Class diagram, but with restrictions and extensions defined for SysML. This diagram defines the structure of the system by blocks, the operations that can be executed by the block, as well as the relationships between blocks, such as associations, generalizations and dependencies (SysML, 2006). Blocks describe specific types of components, connections between components, and the way these elements are combined to define the complete system. SysML Blocks can be used throughout all phases of system specification and design. The SysML Blocks diagram can model either the logical or physical decomposition of a system, as well as software, hardware, or human elements (SysML, 2015).

The definition of a block in SysML can be further detailed by specifying its parts: ports, specifying its interaction points, and connectors, specifying the connections among its parts and ports. Deployment of software to hardware, interaction overview and communication can be represented in the SysML Internal Block diagram. SysML Activity diagram is used to describe the control flow, and inputs and outputs flow between the actions of a system. Within UML Activities, control can only enable actions to start. This is accomplished by providing a model library with a type for control values that are treated like data (SysML, 2015).

3 ARCHITECTURE OF AN AUTOMOTIVE EMBEDDED SYSTEM WITH SysML

Automotive embedded systems can be divided among "vehicle-centric" functional domains, such as power train control, chassis control, and active or passive safety systems, and "passenger centric" functional domains where multimedia/telematics, body/comfort, and human-machine interface can be identified (Zurawski, 2009). The body domain contains functions embedded in a vehicle that are related to the wipers, lights, doors, windows, seats, airbag, and mirrors. These functions have been controlled more and more by software-based systems in past years. In general, they are not subject to stringent performance constraints and, from a safety standpoint, they do not represent a critical part of the system. However, there are certain functions, like an advanced system whose aim is to control access to the vehicle for security, that have to respect hard real-time constraints (Zurawski, 2009).

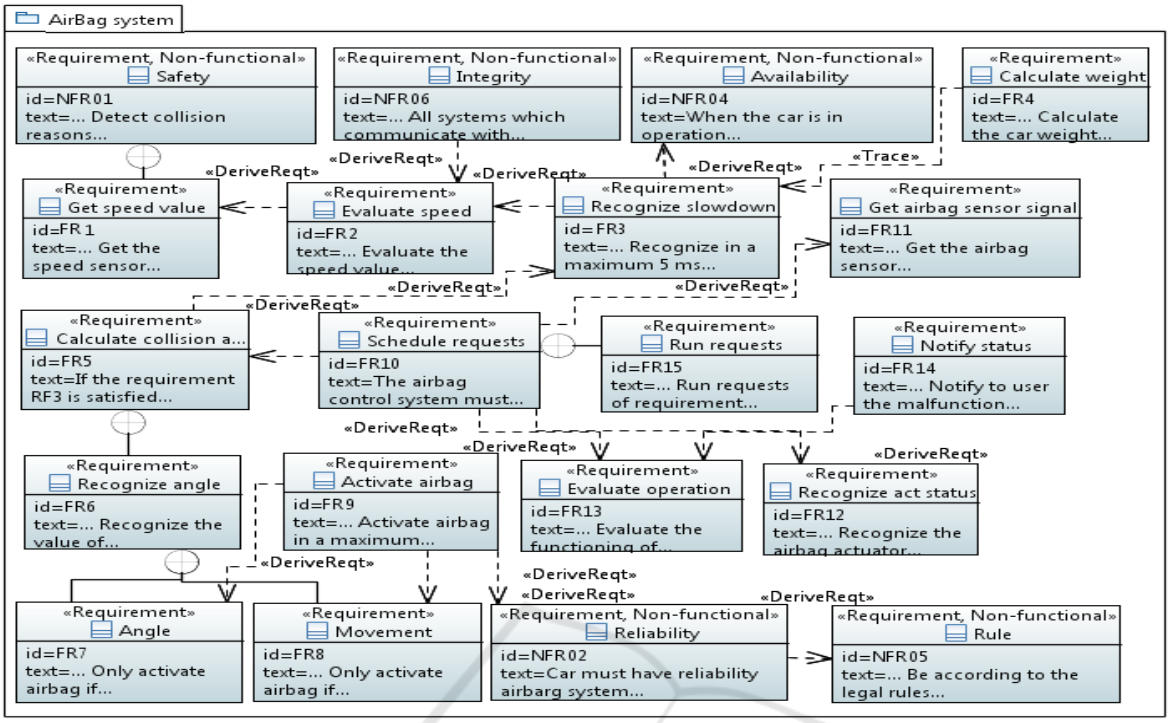

Figure 1: SysML Requirements diagram.

3.1 Airbag Control Example

The Airbag control system is composed by controllers that diagnose collisions through internal sensors, external sensors, through actuators for airbag activation by the diagnosis module. Airbag control is able to evaluate the operation, when the vehicle is turned on. In this section, we first present the example that we use in our modeling approach followed by description of its textual requirements, the SysML Requirements diagram (Figure 1), and SysML Table (Table 1).

Airbag is a safety item able to soften the impact of a possible collision and protect the lives of passengers. Coupled with the safety belt, airbag protects the driver and any passenger against the effects of a frontal impact on the vehicle. Airbag system is not activated in side impacts, rear impacts or rollover. It is activated only if the impact occurs within a maximum angle of 30 degrees with respect to front movement.

Sensors and actuators are incorporated into the system control unit. Sensors recognize a sharp slowdown, identifying when the speed varies in at least 20 km/h in a short time, as in a collision. Actuator aims to determine the moment of airbag triggering. Control unit sends an electrical signal to the igniter that is responsible for inflating the airbag. Within the igniter, substances such as nitrates and ammonia guanidine react and explode instantly. Chemical reaction generates enough nitrogen to fill up the bag in just 30 milliseconds. Airbag begins to empty on impact with the body. Airbag contains a chemical generator gas in a solid state. This gas is stored in a metal chamber. When the control unit receives the sensor signal, an electric current is applied to the bag.

3.2 Set of Airbag Requirements

In this example, according to the characteristics listed before, and results of the detailed domain analysis we conducted, the main requirements for modeling of such complex system are (NFR stands for Non-Functional Requirements, and FR stands for Functional Requirements):

NFR01: The airbag control system must detect collision reasons.

NFR02: The car must have reliability airbag system - tests must guarantee low rate (0.10%) of software failures.

NFR03: The airbag control system must use security protocols in data transmission.

NFR04: When the car is in operation, the airbag control system must be available 99,6 % of time.

NFR05: The airbag control system must be designed according to legal rules (omitted for review).

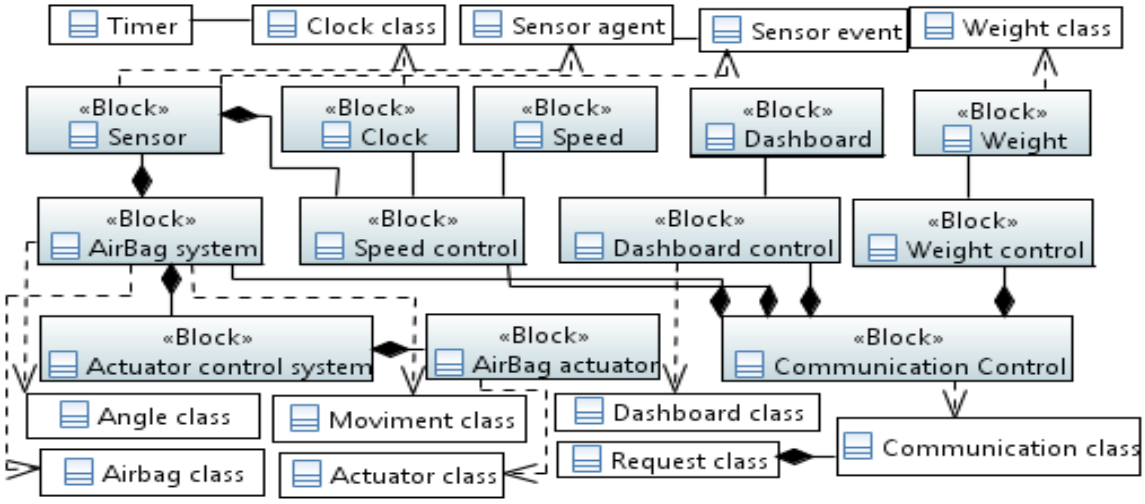

Figure 3: UML Class diagram combined with SysML BDD diagram.

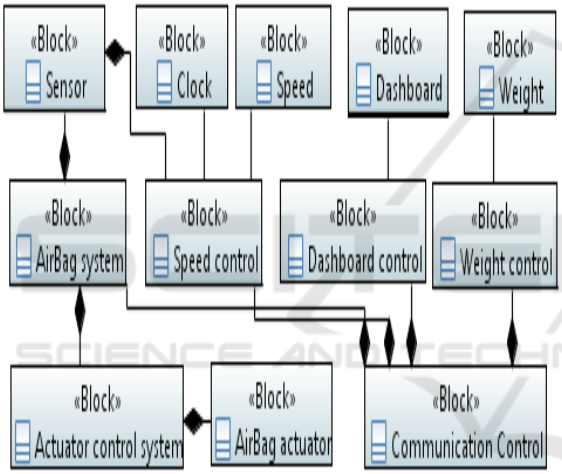

Figure 2: SysML BDD diagram.

NFR06: The airbag control system must be righteous, all systems which communicate with the airbag system must be identified, monitored and blocked from changing data, unless they are authorized.

FR1: The airbag control system must get the speed sensor signal of car every 15 ms.

FR2: The airbag control system must evaluate the speed value received from sensor every 15 ms.

FR3: The airbag control system must recognize in a maximum of 5 ms an abrupt deceleration of at least 20 km/h.

FR4: When the car is in operation, the airbag control system must calculate the car weight plus the passenger weight in a maximum of 5 s.

FR5: If the requirement RF3 is satisfied, then the airbag control system must calculate the collision impact angle in exactly 5 ms.

FR6: The airbag control system must recognize the value of collision impact angle in a maximum of 5 ms.

FR7: The airbag control system must only activate the airbags if the impact angle is lower than 30 degrees.

FR8: The airbag control system must only activate the airbags if the collision impact is at frontal movement.

FR9: The airbag control system must activate the airbags in a maximum of 15 ms after collision impact.

FR10: The airbag control system must schedule simultaneous requests every 15 ms.

FR11: The airbag control system must get the airbag sensor signal every 15 ms.

FR12: The airbag control system must recognize the airbag actuator status every 5 ms.

FR13: The airbag control system must evaluate the functioning of all components every 15 ms.

FR14: The airbag control system must notify to user the malfunction of any light component (with a light on the dashboard) in a maximum of 50 ms.

FR15: The airbag control system must run requests of requirement RF10 every 10 ms.

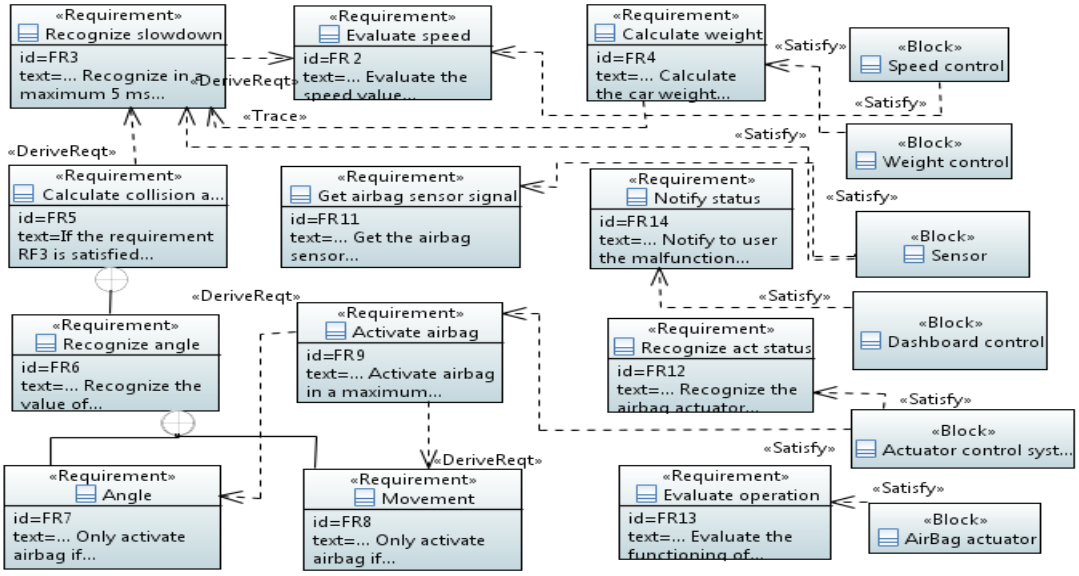

Figure 4: SysML Requirements diagram combined with Sysml BDD diagram.

3.3 Airbag Requirements (Diagram and Table)

The SysML Requirements diagram of the airbag control system is depicted in Fig. 1. SysML also allows the representation of requirements, their properties and relationships in a tabular format. Normally, table fields' are the requirement's ID, name and type. In our proposal, we added fields "derived Req", "derivedFrom Req", "containment Req" (this relationship represents hierarchy) in Table 1.

Requirements diagram refers to the practice of decomposing a complex requirement into simpler, single requirements. In the airbag system, the relationships between requirements were mostly described by ⟨⟨DeriveReq⟩⟩ and containment (represented by $\oplus-$ symbol).

The Derive relationship relates a derived requirement to its source requirement. This typically involves analysis to determine the multiple derived requirements that support a source requirement. The derived requirements generally correspond to requirements at the next level of the system hierarchy. Containment relationship specifies hierarchies between requirements, its use precludes reusing requirements in different contexts since a given model element can only exist in one containment (SysML, 2015).

With the goal to model the non-functional requirements concept, we have also introduced a stereotype named ⟨⟨Non-functional⟩⟩. This stereotype allows to distinguish with others, and facilitates to understand the architecture.

3.4 Airbag BDD Diagram

In our example, we use BDD to model components involved in the airbag system. Specified functions are implemented for secure communications with other components, and with local sensors/actuators. SysML BDD diagram is depicted in Fig. 2. Airbag functions often involve many communications between each other, and consequently, have a complex distributed architecture. These functions are implemented by a main electronic control unit (ECU) named communication control, which supports the reception of requests while the other units perform their own requested actions.

In the Airbag example, 12 related blocks are defined. Relationships between blocks are represented by composition associations. Composition instance is synchronous, because the "Clock" Block controls the time function. If an instance is destroyed, ending execution, the other executions are also terminated. With SysML BDD diagram, hardware, data, and procedures can be modeled. The representation of system architecture can be made by means of blocks, without focusing only on the software structure of each system element, but also on the general structure, including parts of each block, constraints and properties not necessarily related to software. In addition, SysML Blocks are candidates to be refined into one or more UML classes (See Fig. 6) during the software design and implementation phases.

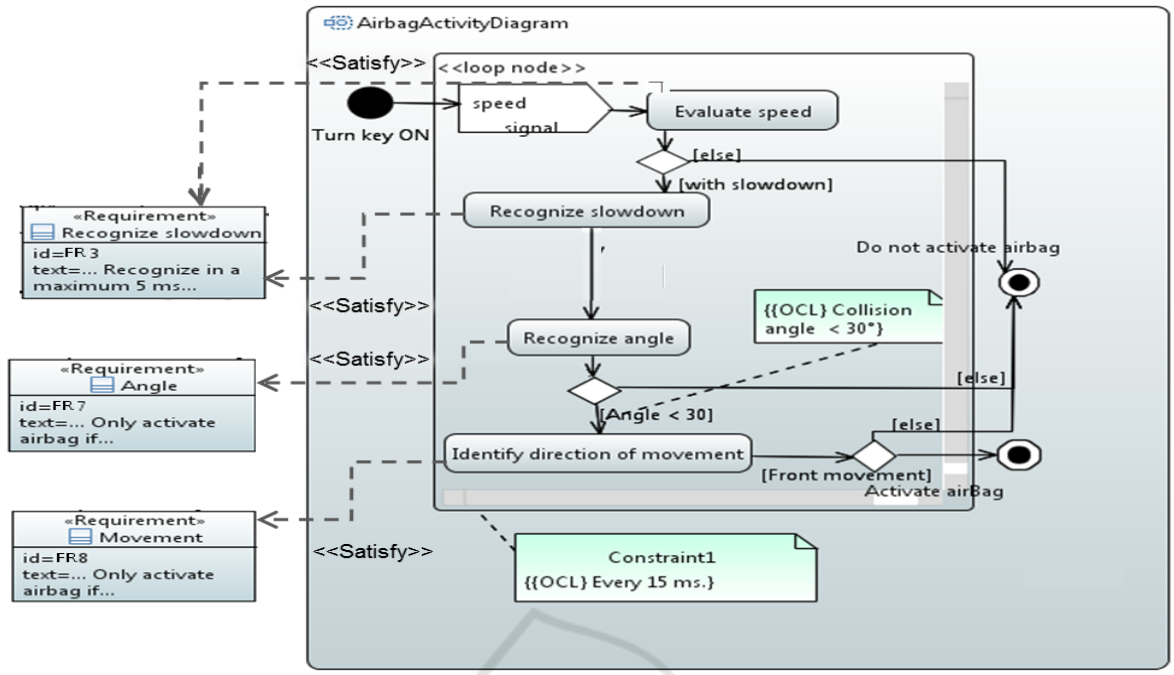

Figure 5: SysML Requirements diagram and SysML Activity diagram.

3.5 SysML Activity diagram

We represent the main Activity diagram of the airbag control system in Fig. 5. Our diagram introduces the activity to activate the airbag. We modeled initial and final activity, decisions node that choose between outgoing flows, iterative loop with ⟨⟨loop node⟩⟩, and constraints for time and angle.

4 PROPOSED TECHNIQUE COMBINING SYSML AND UML

Our approach concentrates on models designed along three architectural views: requirements, structure, and behavior. We use SysML Requirements diagram and Table, illustrated in Fig. 1 and Table 1 for expressing requirements. We model the system structure using SysML BDD diagram, illustrated in Fig. 2. Finally, we use SysML Activity diagram, illustrated in Fig. 5, for capturing behaviors. Our technique is applied to relate the models and to analyze the impact of requirements in both software and system design of an embedded system.

In Table 1, it is possible to analyze how a change in one requirement will impact in the others (derived Req, derived from Req, and containment Req). To perceive this impact, it is necessary to comprehend the relationship of requirements.

Derive relationship relates a derived requirement to its source requirement. This typically involves analysis to determine the multiple derived requirements that support a source requirement. Derived requirements generally correspond to requirements at the next level of the system hierarchy (SysML, 2015). A composite requirement can contain subrequirements, which describes a requirements hierarchy tree. This relationship enables a complex requirement to be decomposed into its containing child requirements (SysML, 2015).

When SysML Blocks and Requirements are related, the design of a system can be analyzed, as shown in diagram in Fig. 4. This diagram is not complete due to lack of paper space limit. Thus, it is possible to identify which requirements are associated with a particular Block, and to identify which requirement belongs to which block. Moreover, relationship satisfy inside this diagram describes how a Block satisfies one or more Requirements. It represents a dependency relationship between a Requirement and a Block.

We combine UML Classes with SysML Blocks by means of the dependency relationship in Fig. 3. Dependency does not need to be labeled, because it is a simple relationship between block and class. The main reason for this choice, to model components with SysML Blocks and UML Class, is because this technique provides a simple way to map systems elements to software elements. Besides, knowing the

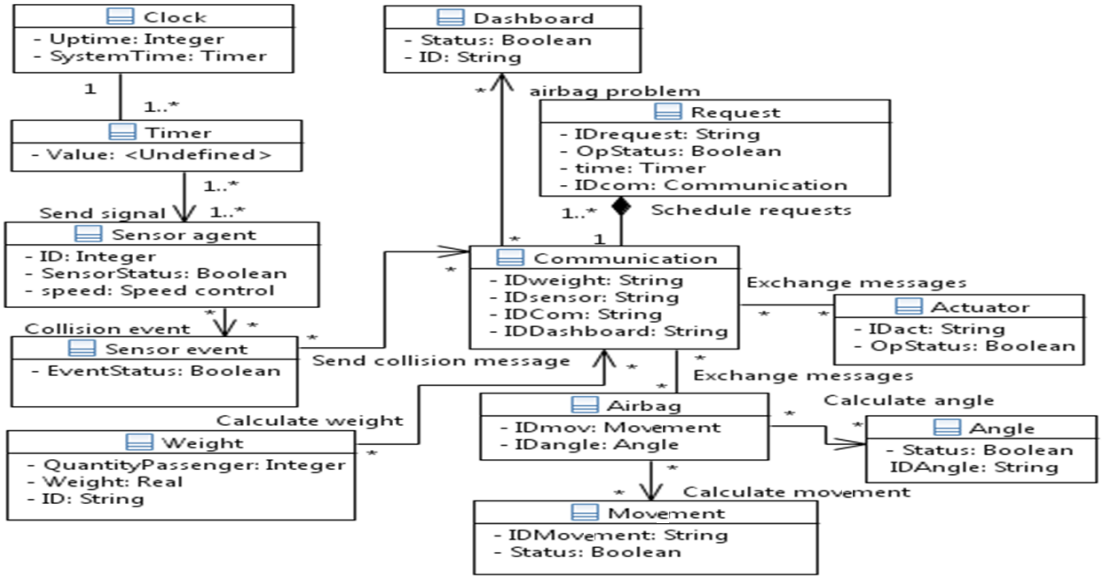

Figure 6: UML Class diagram.

mapping from system elements to software elements may bring together the work of systems and software engineers (Melo and Soares, 2014).

By combining SysML Activity diagram with SysML Requirements diagram, as depicted in Fig. 5, it is possible to analyze how a change in one requirement will affect the activity developed, what actions will be satisfied by a requirement, and also which actions will be affected if it occurs a change in this requirement.

SysML Requirements diagram allows to map a requirement to other design models, or other diagrams. Relationships between requirements, and their classifications, are also represented. For instance, when a requirement is modified, it is possible to trace which requirements and models are affected, and how they are going to be affected.

SysML Blocks are mapped into one or more UML Classes during the software design and implementation phases. This mapping is a modeling part that does not have strict rules (Authors reference) (Melo and Soares, 2014).

5 RELATED WORKS

Recently, the combination of languages with SysML has been investigated for embedded and complex systems in order to support a systems design. The authors of paper (Melo and Soares, 2014) combined the SysML Block diagram with the UML Class diagram to design the structural view of a software-intensive system architecture. An experiment is presented in paper (Vogel-Heuser et al., 2014) regarding UML and SysML on aspects of usability. The intention was to show the increasing of efficiency and quality of software in the development process. A model-driven requirement engineering approach for the embedded software domain is described in (Marques et al., 2014). The approach is based on UML, MARTE and SysML notations, which are integrated to improve requirements specification and traceability. Finally, in paper (Khan et al., 2015) a unified approach is proposed for system design and its verification based on SysML, MARTE, CCSL and SystemVerilog to help hardware engineers in the verification process. Nevertheless, combining three modeling languages (SysML, MARTE, CCSL) plus a hardware language (SystemVerilog) can be obscure for engineers, needs additional efforts, time, and cost to manage the process. In addition, the authors could have explored more SysML diagrams.

In contrast with other papers, our approach has focus to combine UML and SysML, a UML profile, in order to communicate the same project with different stakeholders involved. Our approach shows traceability between SysML Requirements, between structural design with SysML Requirements and SysML Blocks, with SysML Blocks and UML Classes, and also describes behavioral design with UML/SysML Activity diagram and SysML Requirements diagram. None of these related articles showed all these characteristics. Some articles focus only on requirements traceability, others on model synchronization, and so on. In addition, we present languages with the same semantics, this aspect is important to facilitate the

process of learning. To the best of our knowledge, we have only found the works (Vogel-Heuser et al., 2014) and (Melo and Soares, 2014) that combine SysML and UML in the last five years. Concerning the combination of languages, the authors show only the concern of finding the correct relationship between UML and SysML diagrams for a particular focus.

6 CONCLUSION

In the technique proposed in this paper, we combine SysML with UML for describing the architecture of embedded systems, with the main purpose of considering all advantages that both languages combined have. We present a work motivated by reality (Zurawski, 2009). Descriptions and concepts about airbag system came mainly from there.

SysML Requirements diagram and SysML Tables are recognized as useful in activities of Requirements Engineering by many authors. A SysML Requirement can appear on other UML/SysML diagrams to show its relationship to design. The relationships between requirements can improve the specification of systems, as they can be used to model, document and analyze requirements. SysML Requirements diagram support traceability between requirements and modeling of other types of requirements besides the functional ones. We observe the advantages of the SysML modeling language in Requirements Engineering, recognize SysML general characteristics, and expand SysML concepts to model system architecture of a real-time embedded system. Our proposed technique integrates SysML with UML in order to include SysML with software modeling elements, such as UML Classes.

In this paper, we describe how we combined two modeling languages to provide a common modeling language for specifying embedded systems at different abstraction levels. The Airbag system is considered as an example inspired by reality. In comparison with the other items of body comfort, airbag control system was chosen because it has more interaction with other car's components, has more defined requirements, activities in real-time, and represents a car security item. Besides, to the best of our knowledge, we have not find research related to airbag architecture modeling with SysML or other language.

Our technique uses MOF conformance modeling languages (SysML and UML) to help engineers and software architects to communicate and to develop optimized system solutions. We understand that UML and SysML are well-known and generally preferred in software industry because they reduce training costs, reduce the learning time gap, and have adequate tool support.

ACKNOWLEDGEMENTS

The authors would like to thank the Brazilian research agency CNPq (grant 445500/2014-0) for financial support.

REFERENCES

Gardazi, S. U. et al. (2009). Survey of Software Architecture Description and Usage in Software Industry of Pakistan. In *International Conference on Emerging Technologies*, pages 395–402.

Jouault, F. and Delatour, J. (2014). Towards Fixing Sketchy UML Models by Leveraging Textual Notations: Application to Real-Time Embedded Systems. In *Proceedings of the 14th International Workshop on OCL and Textual Modelling co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 30, 2014.*, pages 73–82.

Khan, A. M., Mallet, F., and Rashid, M. (2015). Modeling SystemVerilog Assertions using SysML and CCSL. In *Electronic System Level Synthesis Conference*.

Marques, M. R. S., Siegert, E., and Brisolara, L. (2014). Integrating UML, MARTE and SysML to Improve Requirements Specification and Traceability in the Embedded Domain. In *12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 176–181.

Melo, M. d. S. and Soares, M. S. (2014). Model-driven Structural Design of Software-intensive Systems Using SysML Blocks and UML Classes. In *Proceedings of the International Conference on Enterprise Information Systems*, volume 2, pages 193–200.

Reggio, G., Leotta, M., Ricca, F., and Clerissi, D. (2015). *What Are the Used UML Diagram Constructs? A Document and Tool Analysis Study Covering Activity and Use Case Diagrams*.

Soares, M. S. and Vrancken, J. Model-Driven User Requirements Specification Using SysML. *Journal of Software*, 3(6):57–68.

SysML, M. T. (2006). Systems Modeling Language (SysML) Specification. *OMG document: 2006-03-01*.

SysML, O. (2015). OMG Systems Modeling Language (OMG SysML).

Vogel-Heuser, B. et al. (2014). Usability Experiments to Evaluate UML/SysML-based Model Driven Software Engineering Notations for Logic Control in Manufacturing Automation. *Journal of Software Engineering and Applications*, 7(11):943.

Zurawski, R. (2009). *Embedded Systems Handbook, 2-Volume Set*. CRC Press, Inc.