Developing Dependable Automotive Embedded Systems using the EAST-ADL; representing continuous time systems in SysML

Carl-Johan Sjöstedt[1], De-Jiu Chen[1], Phillipe Cuenot[2], Patrick Frey[3], Rolf Johansson[4], Henrik Lönn[5], David Servat[6], Martin Törngren[1]

[1] Royal Institute of Technology, SE-100 44 Stockholm, Sweden
[2] Siemens VDO, 1 Avenue Paul Ourliac, BP 1149 31036 Toulouse Cedex 1, France
[3] ETAS GmbH, Borsigstr. 14, 70469 Stuttgart, Germany
[4] Mentor Graphics, Theres Svenssons Gata 15, SE-417 55 Gothenburg, Sweden
[5] Volvo Technology Corporation, Electronics and Software, SE-405 08 Gothenburg, Sweden
[6] CEA List, Comtmissariat à l'Énergie Atomique Saclay, F-91191 Gif sur Yvette Cedex, France
[1] {carlj, chen, martin}@md.kth.se
[2] philippe.cuenot@siemens.com
[3] patrick.frey@etas.de
[4] rolf_johansson@mentor.com
[5] henrik.lonn@volvo.com
[6] david.servat@cea.fr

Abstract. The architectural description language for automotive embedded systems EAST-ADL is presented in this paper. The aim of the EAST-ADL language is to provide a comprehensive systems modeling approach as a means to keep the engineering information within one structure. This facilitates systems integration and enables consistent systems analysis. The EAST-ADL encompasses structural information at different abstraction levels, requirements and variability modeling. The EAST-ADL is implemented as a UML2 profile and is harmonized with AUTOSAR and a subset of SysML. Currently, different ways to model behavior natively in the language are investigated. An approach for using SysML parametric diagrams to describe equations in composed physical systems is proposed. An example system is modeled and discussed. It is highlighted that parametric diagrams lacks support for separation between effort and flow variables, and why this separation would be desired in order to model composed physical systems. An alternative approach by use of SysML activity diagrams is also discussed.

Keywords: EAST-ADL, automotive embedded systems, UML, SysML, parametric diagrams, physical modeling, continuous systems, Modelica

1 Introduction and goals

New functionality in automotive systems is increasingly realized by software and electronics. A system level function, such as an adaptive cruise controller will then be partitioned into functions that are realized by software and electronics (the vehicle embedded system), and other functions realized in mechanical subsystems. The complexity of embedded systems calls for a more rigorous approach to system development compared to current state of practice. A critical issue is the management of the engineering information that defines the embedded system. This issue should be understood in the light of current development practice which is characterized by the involvement of a very large number of specialists/groups/companies: all participants are working on the same system but using different tools, models, information formats, and subsets of the complete information. In current practices, integration of artifacts from different parties takes place at a very late stage of the development process where electronic control units are integrated into the overall embedded system of a vehicle. There is a need to shift this hardware level integration to model level integration [2]. Model based development has the potential to improve the cost efficiency of the products including their quality.

In this paper, we will discuss one aspect of model based development, the modeling of the environment to the developed system. Environment models serve several purposes in an architecture description language (ADL). An environment model defines implicitly the context and relevant use of the systems and functions in the embedded systems architecture. Validation activities such as simulation, interface consistency, formal verification all benefit from an environment model. The environment of an automotive embedded system may include all kinds of systems and behaviors that are part of the embedded system itself. The focus here is on physical continuous time systems. In particular we investigate the representation of continuous systems in SysML parametric diagrams, using Modelica [6] compatible constructs.

The presented approach is a part of an effort to refine an architecture description language for automotive embedded systems. An initial version of this language, EAST-ADL, was developed in the EAST-EEA project [19]. Further work on the the language is pursued in the ATESST project [17]. For other aspects of the EAST-ADL, see [3].

2 Overview of the EAST-ADL

The EAST-ADL is intended to support the development of automotive embedded software by capturing all the related engineering information. The scope is the embedded system (hardware and software) of a vehicle and its environment. The EAST-ADL system model is organized in parts representing different levels of abstraction and thus reflects different views and details of the architecture. The levels implicitly reflect different stages of an engineering process, but the detailed process definition is company specific.

The EAST-ADL language constructs support:

- vehicle feature modeling including concepts to support product families
- concepts for defining variability in all parts of a model
- vehicle environment modeling to define context and perform validation

- structural and behavioral modeling of software and hardware entities in the context of distributed systems.
- requirements modeling and tracing with all modeling entities
- other information part of the system description, such as a definition of component timing and failure modes, necessary for design space exploration and system verification purposes

The language is structured in five abstraction levels (see Fig. 1), each with corresponding environment system representation (in parenthesis):

- *Operational Level* supporting final binary software deployment (operational architecture)
- *Implementation Level* describing reusable code (platform independent) and AUTOSAR compliant software and system configuration for hardware deployment (implementation architecture)
- *Design Level* for detailed functional definition of software including elementary decomposition (design architecture)
- *Analysis Level* for abstract functional definition of features in system context (analysis architecture)
- *Vehicle Level* for elaboration of electronic features (vehicle feature model)

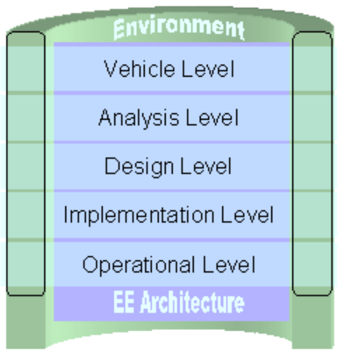

Fig. 1. EAST-ADL language abstractions. Note that the environment model spans all abstraction levels, and that requirements and variability constructs apply to modeling elements regardless of abstraction level.

2.1 EAST-ADL definition, implementation and relation to other languages

In defining the EAST-ADL language, a two step procedure is adopted. The final language is implemented as a UML2 profile. A domain model is first defined, capturing only the domain specific needs of the language. The domain model thus represents the meta-model, the language definition. Basic concepts of UML are used for this purpose, such as classes, compositions and associations. Based on the Domain model, a UML2 profile implementation, the “UML viewpoint” is defined with stereotypes, tags and constraints. This implementation is delivered as an XMI file ready for use in UML2 tools. As a proof-of-concept, an Eclipse based prototype tool with supporting analysis features, Papyrus [20], has been implemented. The EAST-ADL language also incorporates relevant aspects from SysML [14] and MARTE [9]. SysML is a modeling language that supports the specification, analysis, design, verification and validation of systems which may include hardware, software, information, processes, personnel, and facilities. SysML is defined in terms of a UML2 profile. MARTE is an ongoing effort to define a UML profile for Modeling and Analysis of Real-Time and Embedded systems, initiated to overcome the UML limitations of modeling such systems. The EAST-ADL is also being harmonized with the new automotive domain standardization AUTOSAR [18]. AUTOSAR focuses mainly on the implementation level of abstraction, whereas the EAST-ADL supports the overall comprehensive systems modeling.

Inspiration in the development of the EAST-ADL is also gathered from the SAE Architecture and Analysis Description Language (AADL) [16] and safety standards such as the ISO 26262 Functional Safety (committed draft planned for beginning 2008).

Considering the multitude of languages that in different ways address embedded systems, a relevant question is how the EAST-ADL relates to other modeling language efforts. This was partly elaborated in the previous text, but the main reasons for introducing "yet another language" are summarized here for clarity:

- EAST-ADL vs. UML: UML is a general modeling language for software engineering, which contains no specifics for automotive embedded systems. The EAST-ADL provides a tailoring of UML2 through a profile dedicated for such systems.
- EAST-ADL vs. SysML: SysML is a UML2 profile for systems engineering. EAST-ADL incorporates several SysML concepts and specializes them as needed for automotive embedded systems.
- EAST-ADL vs. AUTOSAR: AUTOSAR focuses on software and hardware implementation. The EAST-ADL complements AUTOSAR with e.g. functional specifications and requirements and reuses AUTOSAR concepts for the implementation level abstractions.
- Why not proven proprietary tools and languages such as MATLAB/Simulink [13], ASCET [5] or Modelica? The very fragmentation into multiple domain/discipline tools that target different aspects of the system is a key driver for developing the EAST-ADL. The EAST-ADL language provides an information structure for the engineering data required as a basis for automotive embedded systems development. In the ATESST project, interfaces in terms of model transformations and tool interfaces for the prototype tool Papyrus are developed to domain tools/languages.
- Why not information management tools such as product data management tools (PDM)? Such tools lack an information model for automotive embedded systems and the connections to external domain tools. The EAST-ADL domain model could be used as a basis for information management in existing PDM-like tools. Moreover, the use of UML2 allows native behavior to be defined. The fact that UML2 is a standard allows the EAST-ADL to be used with several UML tools.

2.2 Behavior modeling approach in EAST-ADL

With respect to behavior the goal of the EAST-ADL is to provide native behavior descriptions, for primitive components, as well as for compositions. Native behavior is useful for example to describe the desired overall behavior of the vehicle systems as well as the environment. The objective includes a native behavioral notation that allows simulation and verification within the defined system model, but also concerns the integration of external tools, as part of today’s industrial practice for designing behavior algorithms of vehicle applications. Integration here refers to the ability to import/export models to and from external tools based on model transformations.

3 Physical systems modeling in SysML

The overall goal is to find a representation of continuous systems in SysML to be used in the EAST-ADL. Parametric diagrams and Modelica models have many things in common (equation based, acausal, modular etc.), so the hypothesis was to see how they relate to each other. The approach was to make a proof-of-concept description of a Modelica model using native SysML constructs.

3.1 Modeling physical systems

To get a complete description of the system, not only the embedded system needs to be modeled, but also the environment it interacts with. In control theory this is referred to as the plant model. One possibility in the EAST-ADL is to rely on legacy tools, most notably Simulink and ASCET, for defining plant behavior. The functions that compose the environment model define the structure, and a link to a behavioral definition in the external tool is provided for each of these. The complete behavior of the environment model is the result of the composition of the parts. Alternatively, and this is elaborated below, an equation-based behavioral definition is used for environment model behavior. This behavior is defined as a part of the EAST-ADL, and would be understood by EAST-ADL compliant tools.

A typical plant model is described by differential equations, but can also include hybrid systems. Example of the latter kind is a gearbox, or the state-transition between different properties of a road, e.g. from ice to water. The interface between the plant and the embedded system is in form of inputs (actuators act as inputs to the plant) and outputs (sensors act as outputs from the plant to the control system). In the implementation, this will be realized as an interface between a discrete embedded system and a continuous “real world”. During modeling and design, this interface could be continuous to continuous at the Analysis Level (from Fig. 1), or discrete to discrete when modeling and simulating a plant model.

Just like the embedded systems, plant models can be described at different levels of abstraction, see Fig. 2. These abstraction levels are adapted from [12], and have been proven useful when discussing how models could be described in a uniform way, e.g.

are the models on the same abstraction level, and can they be integrated on that level? There are also levels orthogonal to this, e.g. more or less advanced models, more or less validated models etc. These abstraction levels are not directly correlated to the EAST-ADL abstraction levels of the embedded system.

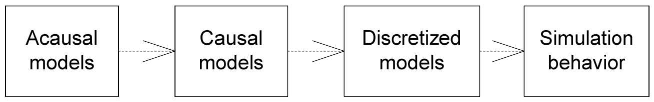

Fig. 2. Different levels of abstractions of environment models

Acausal models. At the highest level of abstraction for environment models, there are acausal models, where models are described by one or several differential or algebraic equations, possibly combined with state machines to model hybrid systems. The acausal Modelica components are on this level of abstraction, where equations and their relation to the outside world are modeled. Acausal models are generally more flexible and reusable than models at a lower abstraction level [6].

Causal models. At the causal level of abstraction, it is defined what is input, and what is output inside the system, and between components in a composed system. Typical causal models include bond graph representations [11], or block diagrams, like continuous Simulink models.

Time-discretized models. To solve a differential equation numerically it is typically discretized in time. A discretized model is an algorithmic representation in the sense that it generates a defined output for a certain input and internal state. A model can be discretized in different ways, e.g. using forward/backward Euler.

Simulation behavior. To perform the calculations of the discretized models, a solver and a scheduler is needed as part of the simulation engine. The simulation engine can decide the time-step, execution order, triggering, communication, etc. of the model. Typical numerical tools to solve differential equations are Simulink and ASCET [5]. Here, the simulation behavior of the continuous time model can be described using the same Model Of Computation (MOC) as for the model components of the embedded system.

Proceeding to lower abstraction levels means that the model gets more sophisticated, in the sense that the model can produce simulation results. On the other hand, the models get more specialized and differ more from the original system. Moreover, numerical errors could be introduced at all transformations. Tools and methods could use many of these abstraction levels, and hide some from the user. For example, a continuous Simulink model is modeled as a causal model. The choice of solver provides a way of (indirectly) discretizing the model since it implies a discrete-time implementation through the simulation behavior built into the tool. The Dymola [4] tool makes a point of taking the equations from the highest abstraction level to the lowest, while ASCET uses models close to the lowest level, and letting the user have control over the real-time hardware-in-the-loop simulation.

3.2 SysML parametric diagrams

SysML consists of four behavioral and five structural diagrams. Seven of these diagrams are partially reused from UML, while two are new: the requirement diagrams and the parametric diagrams [14]. The parametric diagrams' conceptual foundation is the composable objects representation (COBs), developed at Georgia Institute of Tecnology [10], the academic partner of the SysML team [14]. COBs provide five basic views of a system, Shape Schematic, Relations, Constraint Schematic, Lexical COB structure and Subsystem, of which Relations, Constraint Schematic and Subsystems have equivalent representation in parametric diagrams.

Parametric diagrams describe constraints between variables, like equations, and how they are related to each other. The SysML specification gives an example of Newtons equation, which can be modeled in continuous time [14]. The constraints are acausal, and by combining many modular subsystems, acausal relationships for a large system can be achieved. In addition, state machines can tell which equations to be used in the parametric diagrams, to be able to describe hybrid systems.

4 Investigation of an example system

As an example system an electrical circuit from [6] was chosen. It was chosen since the Modelica representation is well-described in this reference, and since it has sufficient complexity: different dynamical features and parallel branches. The model also highlights that different causality is needed for the resistors in the two branches.

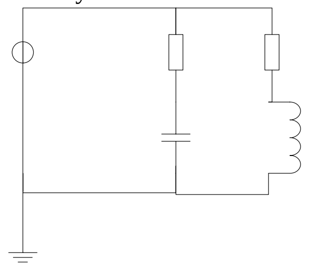

Fig. 3. The example electrical circuit. An alternating voltage source is applied to a circuit containing two resistors, one capacitance and one inductor.

4.1 Definition of a component with SysML

An electrical component is specified using the two-pin class, where general equations common for many electrical components are stored. These equations are also linked together in a parametric diagram.

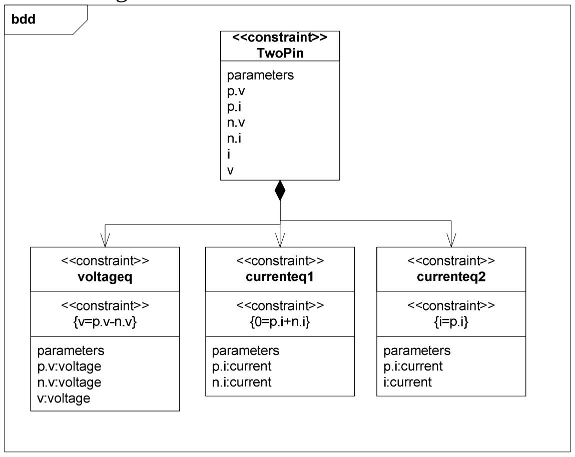

Fig. 4. Constraint definition of the electrical TwoPin Modelica class within a SysML block definition diagram (bdd). There is a separate parametric diagram to show the relation between the equations.

This diagram is also linked in the parametric diagram of the resistor, where variables link to the FlowPorts of the resistor. The component is defined as a SysML block, where bidirectional FlowPorts are used to represent connections. The capacitor, inductance, ACSource etc. can be defined in a similar way.

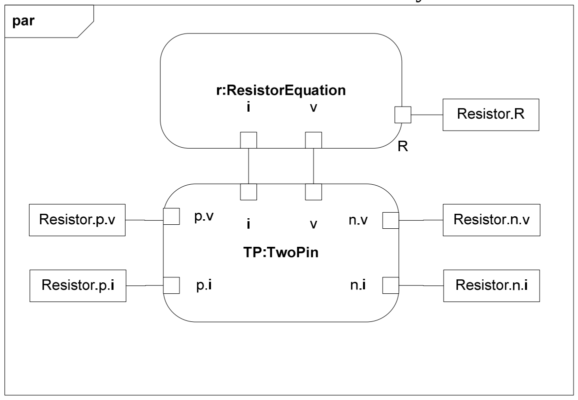

Fig. 5. SysML parametric (par) diagram of the resistor. The Resistor.x.y variables are linked to the FlowPorts of the resistor, and the Resistor.R is a property of the Resistor.

4.2 Composing a system

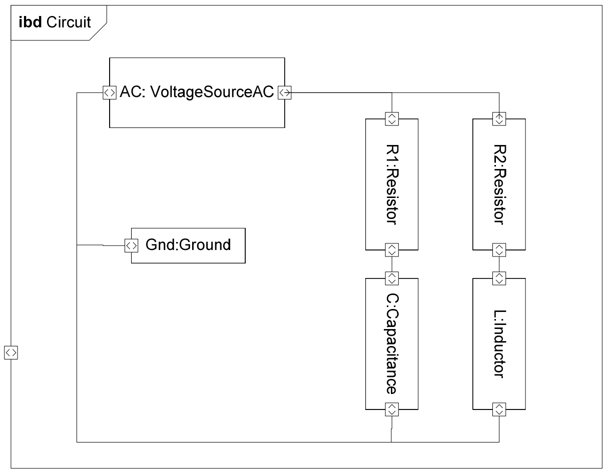

Fig. 6. Internal block diagram (ibd) of the circuit. SysML ItemFlow is specified between components, and have the same function as connectors in Modelica

The example system is composed using instances of the components defined in the previous section. The parameters can then be assigned values. In a first attempt, ItemFlow is specified between the components to be connected, used the same way as connectors would be used in Modelica. ItemFlow is a SysML stereotype describing the flow of items across a connector or an association. This is an intuitive way to connect the components, since it is related to the physical layout of the circuit. The problem is that when calculating the current from these parametrics, a wrong result will occur: The current will be the same everywhere in the system, which is not possible due to Kirchoff's current law. In Modelica this is handled by defining the current as a "sum to zero"-variable, and the voltage as an "equality"-variable. In a connection having many branches, the equality variable will have the same value for all branches, while the "sum to zero" variables will be summed to zero. Here, a workaround has to be made. A possible solution is to include a parallel flow split component, which is shown in Fig. 7. One of the two-pin current equations must also be changed from `0=p.i+n.i` to `p.i=n.i`.

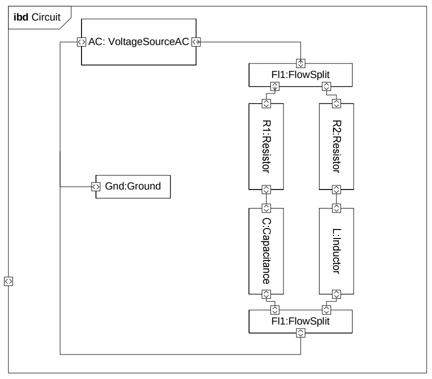

Fig. 7. An acausal representation of the system, where the equations are fully described by the underlying parametric diagrams

4.3 Block diagram model expressed as an activity diagram

The system could also be described at the causal level of abstraction. The circuit could be transferred to a bond graph representation. There are systematic approaches for electrical circuits, see for example [8]. Bond graphs could be described using SysML activity diagrams [15]. A bond graph model can also be converted to a block diagram [11], which could be expressed as an activity diagram, as described in [1]. This representation is shown in Fig. 8. To execute this diagram, initial values are needed on both "Add"-actions. When simulating a model in Simulink, the execution order is however typically not from input to output, it is decided by which blocks have direct-feed through and those that have internal state, see [13] for more information on this.

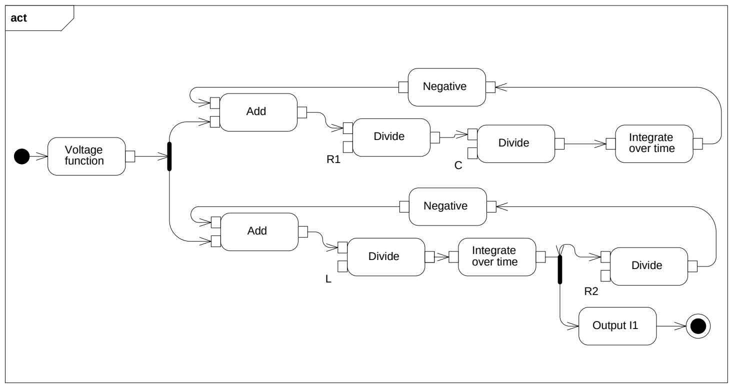

Fig. 8. An activity diagram describing the circuit. This is similar to the Simulink representation, although the sequencing of the actions differ

5 Discussion and Conclusions

An attempt to make modular simulation models of physical systems using SysML has been tested and evaluated. The concepts includes internal block diagrams with attached parametric diagrams, bi-directional FlowPorts, and ItemFlow, The approach is maybe not the intended way to use these diagrams. The intention in SysML is to connect parametric diagrams to each other to generate a composed parametric diagram for the system. Then, the governing equations in this parametric diagram are separated from the structure in the internal block diagrams.

Although parametric diagrams are acausal, they do not contain any separation between effort and flow variables, which is fundamental when modeling physical systems [11]. Both Modelica (using `flow`) [6] and VHDL-AMS (using `across/through`) [7] contain such constructs. The solution of introducing the flow-split block is not an appealing solution; the point of using acausal diagrams is then somehow lost. Another solution would be to introduce stereotypes for flow variables, to show that they are to be summed to zero. A parametric diagram could be manually generated, or in principle automatically generated, based on the properties of the internal block diagram.

The behavior of the composed system could of course also be modeled using a non-modular parametric diagram. An optimized system model of the example system contains seven equations [6], which could easily be modeled using a single parametric diagram.

Using activity diagrams it is possible to model a block model version of the system. The activity diagram is a behavioral diagram, as opposed to the parametric diagrams (which are structural). This is a way to capture the structure of a Simulink model which is the causal representation of the system. This is an interesting path and will be further investigated in the ATESST project.

The compatibility between Modelica and UML representations needs further investigation. The advantage of having a Modelica compatible description is that it could be translated to Modelica models, and then simulated in existing Modelica tools.

Acknowledgements. This work has been carried out within the ATESST project, 2004-026976 in the EC 6th Framework Programme.

References

1. Bock, C., SysML and UML 2 Support for Activity Modeling, Wiley Interscience (2005)
2. Chen, D-J., Törngren, M., Shi, J., Årzen, K-E., Lönn, H., Gérard, S., Strömberg, M., Servat, D.. Model Based Integration in the Development of Embedded Control Systems – a Characterization of Current Research Efforts. In Proc. of the IEEE International Symposium on Computer-Aided Control Systems Design, Technische Universität München, Munich, Germany (2006)
3. Cuenot, P. Chen, D-J., Gérard, S., Lönn, H., Reiser, M-O., Servat, D., Tavakoli Kolagari, R., Törngren, M., Weber, M., Improving Dependability by Using an Architecture

Description Language. Accepted book chapter contribution for the forthcoming book /Architecting Dependable Systems IV/. Editors: Rogerio de Lemos, Cristina Gacek, Alexander Romanovsky. To be published by Springer, Lecture Notes in Computer Science series.

4. Dymola user manual, Dynasim AB, Lund, Sweden (2005)
5. ETAS Product and Service Catalog 2006/2007, ETAS GmbH, Stuttgart, Germany (2007)
6. Fritzson, P., Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, IEEE Press, Wiley-Interscience (2004)
7. Heinkeel, U. et al., The VHDL Reference, John Wiley, England (2000)
8. Ljung, L. & Glad, T., Modellbygge och simulering, Studentlitteratur, Lund, Sweden (1991)
9. Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), RFP. (2005)
10. Peak, R.S., Burkhart, R.M., Friedenthal, S.A., Wilson, M.W., Bajaj, M., Kim, I., Simulation-Based Design Using SysML—Part 1: A Parametrics Primer. INCOSE Intl. Symposium, San Diego (2007)
11. Rosenberg, D.L., Margolis, D.L., Karnopp, D.C, System dynamics : modeling and simulation of mechatronic systems, Wiley, New York, N.Y. (2000)
12. Sen, S., Vangheluwe, H., Multi-Domain Physical System Modeling and Control Based on Meta-Modeling and Graph Rewriting, proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, Munich, Germany (2006)
13. Simulink user manual, MATLAB release 2006b, Mathworks (2006)
14. SysML Specification v1.0, SysML partners (2006)
15. Turki, S. & Soriano, T., A SysML extension for Bond Graphs support, 5th International Conference on Technology and Automation (ICTA'05) (2005)
16. www.aadl.info
17. www.atesst.org
18. www.autosar.org
19. www.east-eaa.net
20. www.papyrusuml.org