# Increasing Reliability of Embedded Systems in a SysML Centered MBSE Process: Application to LEA Project

Robin Cressent, Pierre David, Vincent Idasiak, Frédéric Kratz

# Increasing Reliability of Embedded Systems in a SysML Centered MBSE Process: Application to LEA Project

Robin CRESSENT, Institut PRISME – ENSIB, robin.cressent@ensi-bourges.fr
Pierre DAVID, Heudiasyc UMR 6599 - UTC, pierre.david@hds.utc.fr
Vincent IDASIAK, Institut PRISME – ENSIB, vincent.idasiak@ensi-bourges.fr
Frédéric KRATZ, Institut PRISME – ENSIB, frederic.kratz@ensi-bourges.fr

***Abstract*- The article presents a new method relying on Model-Based System Engineering concepts in order to efficiently deal with real-time and dependability constraints, throughout the design process of embedded systems. The method is described and its use is exemplified on the LEA project shared by MBDA and the PRISME Institute. The method is built in a SysML-centric context, and proposes to merge current efficient methods for safety, reliability and real time constraints analysis. In particular, the AltaRica Data Flow and AADL languages are integrated in the framework supporting the method.**

## I. INTRODUCTION

The LEA flight test program was started in 2003 by MBDA-France and ONERA to address the key issue of the aeropropulsive balance of a dual-mode ramjet powered vehicle in the range Mach 4 to 8. A development methodology has been defined for such type of vehicle, together with the numerical and experimental tools enhancement to enable predicting the flight performances with suitable accuracy. This methodology is now being applied to minimal size experimental vehicle, called LEA, which has passed the preliminary design review in 2006, and the critical design review in 2009. Finally, several flight tests will be performed at the end of the program to validate the quality of performance prediction. Four flights are planned, and will be performed between 2013 and 2014, operated from Russian test range and using Russian hardware for initial acceleration. Our team is charged of specifying, designing, testing and validating the embedded system, which must control the flight from the launching of the craft to the final crash. Moreover, the embedded system must control the dual-mode ramjet carburetion process and some safety functionalities, like the auto-testing function for automating GO/NoGO decision before launch or the detection of separation from booster and the ignition order. Since three years, we develop around SysML some tools to merge engineering process to safety studies for critical systems [1,2]. The LEA project gives us an experimental platform to adapt our work on embedded system.

After a large presentation of the method developed and its issues, we detail the tools we design and develop to automated the method and take into account the specific point of view introduced by embedded system, the software architecture design and its temporal studies. In the final part, we discuss interesting results like how SysML and AADL models helped us to choose, in the first design stage, the accurate hardware and software architecture thanks to FMEA studies, SysML Parametric Diagrams and SysML Internal Block Diagrams.

## II. THE MÉDISIS METHOD

Nowadays the Model Based System Engineering (MBSE) paradigm is becoming the predominant concept used for System Engineering (SE) [3]. The main idea brought by this practice, is to enhance the design process of complex systems and making it more reliable, by organizing activities through formalized representations of the systems called models. The Model Based representations allow to obtain more consistent, traceable, coherent, reusable and expressive views of the system to be developed, helping the management and realization of its design process.

Since several years multiple SE methods have been created, they describe how using some modeling approaches to carry the system design tasks from stakeholders needs capture to system validation [4]. Each method focuses on employing a specific modeling langue and is used to conduct the tasks defined by SE processes. Those lasts are standardized in various norms tackling diverse class of systems or applicative domains. Few of these methods include strategies to deal with safety and reliability related design activities required by SE processes as IEC 61508 [5] (and its derivations) or the future ISO 26262 [6]. Therefore, we focused our contributions to MBSE on defining a method to improve the realization of reliability analysis during the SE process and its early design phases. This method introduced in several publications [1,2], is called MéDISIS. This method is related to the use of SysML [7]. We assume that inputs models are expressed in SysML and we propose to build a repository registering and managing the knowledge raised by the activities per-

formed in a structure modeled in this language.

The major assumption that was made for the constitution of MéDISIS, was to consider that the method used by the designer to construct the functional model of the system is let totally free. This assumption was taken to make it possible to integrate MéDISIS to others MBSE methods tackling the other tasks of system development.

MéDISIS proposes a deductive and iterative approach aiming at facilitating crucial reliability analysis and enhancing the use of the diverse tools and languages used for dysfunctional behavior validation. MéDISIS includes the following steps also depicted on figure 1:

- Deduction of the dysfunctional behavior with an FMEA, identification of the impacted requirements.
- Construction of a model integrating functional and dysfunctional behaviors with a formal language.
- Analysis and quantification of dysfunctional behavior.

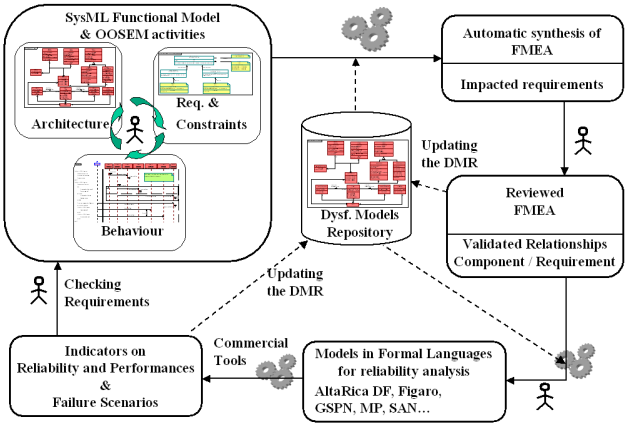


Figure 1. MéDISIS overview

To perform the steps of MéDISIS several tools and analysis routines have been defined to support each phase and optimize the rapidity and quality of the reliability studies. These developments are made to construct a complete System Development Environment (SDE) supporting MéDISIS and system design. Analysis techniques and a tool have defined to support FMEA realization. The results of this study raise the dysfunctional behavior of each component. They are capitalized in a dysfunctional models repository and reused to construct a formal representation of the system using the AltaRica Data Flow [8] language. The construction of this formal model, mandatory for system validation, is also helped by analyses techniques systematizing the creation of this reliability-oriented view. A service to support embedded systems analysis is also available. It proposes to generate AADL [9] models exploitable for real time application studies using Cheddar tool. These three techniques will be summed up in the following section.

## III. BUILDING A SDE FOR MÉDISIS

This section highlights the various techniques and tools employed in MéDISIS to improve the efficiency of reliability studies during design. Developments are made to reduce the gap between the multiple specialized tools utilized for the creation of embedded systems, and others are set up to support analysis techniques as FMEAs.

### A. *Support to the FMEA process*

FMEA is a crucial analysis in the dysfunctional behavior study, since it gives a mean to identify the dysfunctional aspects of each component. Moreover, this technique is fundamental because it organizes the passage from a purely functional view of the system to its unexpected behavior expression using only the designer's description and experts' knowledge. Nevertheless, this founding study for the reliability analysis process suffers of its heaviness of deployment. Authors as [10] and [11] have well summarized the drawbacks of this important but unpopular approach, which they depict as a heavy, error-prone, non-flexible, and often done too late analysis.

Our conviction is that this kind of study can benefit from a Model-Based approach. Therefore, we studied how FMEA could be performed in a MBSE framework, using SysML as a central language. We thus developed a method of functional model analysis for performing FMEA supported by a software tool automating the more tedious parts of the reasoning. Thanks to the Model Based approach, we obtained more consistent FMEAs and improved the lesson-learnt sustainability throughout the system development and the projects concerning a family of products.

The use of FMEA within the MéDISIS framework have been discussed in [1], we will sum up here the techniques involved in the FMEA creation. We propose to construct the FMEA report in three phases, first the functional model is processed, raising the significant points for the FMEA and collecting the lesson-learnt knowledge from a database, then a preliminary FMEA report is automatically produced, finally the FMEA is completed and validated by reliability experts. For the realization of the first point we defined 7 rules for SysML models study employed to identify and highlight the information about the system useful for the FMEA reasoning. For example, component connections, failure propagation paths or probably impacted requirements are tracked. A structure containing for each component, the related neighborhood, behavior and requirements is set up, this one include also the information on the dysfunctional behavior already observed in other applications, stored in the MéDISIS central database. The elements of the structures are then organized in a preliminary FMEA report, proposing for the components the possible failure modes and their potential causes or effects. Those two first phases of the FMEA creation are respectively performed by a model analyzer, using file parsing techniques, and a FMEA table generator organizing the gathered data. The last phase is let to the human interven-

tion of reliability experts, as human reasoning for finalizing an FMEA cannot be performed automatically, except from using artificial intelligence techniques which are not enough confident for this kind of key decisions.

### *B. Connection to formal descriptions with AltaRica DF*

The second support needed in MéDISIS, is the integration of formal means of validation and quantification of the dysfunctional behavior. Many solutions to perform this task are available on the market. Therefore we concentrate on creating bridges between the tools used by functional engineers and those dedicated to reliability studies. We focused on using AltaRica DF language, which is largely spread among reliability engineers and efficiently equipped by solutions as BPA-DAS (product of Dassault Systèmes) [2].

The service is performed in two major steps, which are the translation of the SysML model to obtain the AltaRica DF description of the functional view of the system, and the modeling of the dysfunctional view using the Dysfunctional Model Repository (DMR) built with FMEA results and previous studies lesson-learnt. The first translation is important in order to construct a reliability studies dedicated model consistent with de description of the system that is common to the whole development lifecycle. As SysML and AltaRica DF share an Object-Oriented approach, many translation elements are quite direct. Nevertheless, some divergent declaration philosophies of the two languages, as the treatment of state and flow variables, impose to use more complicated translation rules that we defined. Moreover, the complete automation of the translation is possible only if the semi formal nature of the SysML description is constrained by construction rules of the SysML model like the utilization of expressive allocations between the modeling elements.

The completion of the functional view by the description of the dysfunctional behavior of the components is the occasion to note the benefit of the MéDISIS framework and its DMR centralizing the relevant information for reliability studies. In fact, the data raised by FMEA are added to the AltaRica DF model, thanks to its expression in the DMR. The complete model for formal reliability analysis is thus obtained and then exploited with the market software tools. The metamodel of the DMR have been developed in order to be coherent to the SysML description and to store the needed elements for the construction of the AltaRica DF final model. Therefore the DMR is built in SysML and integrates constructs as statemachines to prepare dysfunctional models creation.

MéDISIS has been constituted as an evolutionary framework aiming at connecting all the needed specialized analysis tools, permitting to assess all system behavior dimensions. It has been augmented with a service for real time constraints considerations exposed in the next paragraphs.

### *C. Support to the Embedded design process*

AADL is a formal and textual language that appeared for the first time in 2004. Its graphical form and other extensions were added in 2006 [14]. The recent revision [9] shows the interest of the community to keep the language up-to-date. The use of AADL gives the opportunity to analyze formally real-time and embedded systems. To reach this aim, the use of a transformation of SysML models to AADL ones is an efficient support. Furthermore, some tools dedicated to AADL exist such as Cheddar [12], which permits to study the scheduling, processor usage, and respect of temporal constraints.

#### 1. *Type* and *implementation*

Each component of the embedded system is described in AADL by its *type*, which refers to the functional interface and its *implementation*, which refers to the inside composition of our component (subcomponents, temporal properties, connections, etc). Besides, *type* and *implementation* can be defined by different persons, each being responsible of different steps in the architecture's design, specifications and detailed design for example.

Each component is part of one of the 3 category of AADL: Execution Platform, Application Software and Composite. Each category is subdivided to reach 10 different categories of components. Each one possesses a graphical notation (see figure 2):

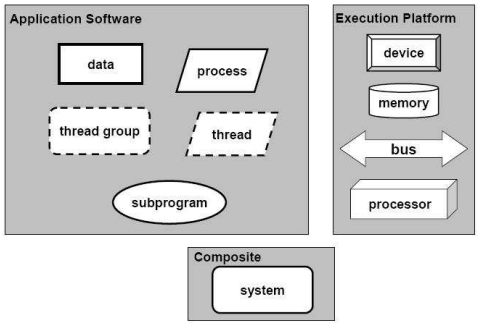


*Figure 2. Graphical notation of the different categories of component in AADL*

Each *type* can be associated to zero, one or more implementations. The example below shows us two *types* of components, the first is associated to *implementations* (see figure 3):

```
system type1
end type1;
system type2
end type2;

system implementation type1.impl1
end type1.impl1;
system implementation type1.impl2
end type1.impl2;
```

*Figure 3. Component's type and implementation example*

The principle of *type* and *implementations* can be compared to the use of BDD and IBD in SysML.

2. *Properties*

Each component possesses properties which serve to characterize the component.

Some properties are part of the language. They are identified by a name and associated to a category of component. For example, *threads* possess their own properties such as execution period, deadline,... (see figure 4).

New properties can be created by the user and associated to one or more category of component. That way, every detail related to the user's need, can be considered, and that is one point that makes AADL very useful.

```
thread thread1
   properties
      Period => 15 ms;
      Deadline => 10 ms;
end thread1;
```

*Figure 4. Thread properties example*

Those *properties* may be found partially by going through the *value properties* in the SysML model.

3. *Port* and *connection*

The description of data and control flow in our AADL model is done using ports and connections. A *port* is point of entry and/or exit of a component, in which data or events can pass through. A connection permits to link two ports. A connection can be made between a port of entry and a port of exit on a same hierarchical level, or between same types port which are in different hierarchical levels. A verification of the conformity of the *types* and directions is made between connected *ports*.

Figure 5 and figure 6 show us a system containing two processes, themselves containing a thread. The group of connected *ports* represented by triangles and the group of connections represented by lines establish the link between the threads.

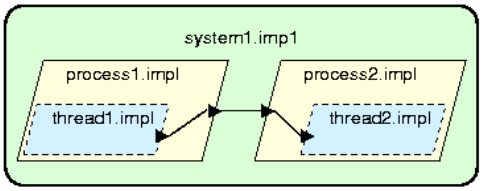


*Figure 5. Connection between two threads*

```
system system1
end system1;

system implementation system1.impl
   subcomponents
      p1: process process1.impl;
      p2: process process2.impl;
   connections
      cn: data port p1.outport -> p2.inport;
end system1.impl;

process process1
   features
      outport: out data port;
end process1;

process implementation process1.impl
   subcomponents
      t1: thread thread1.impl;
   connections
      cn: data port t1.outport -> outport;
end process1.impl;

process process2
   features
      inport: in data port;
end process2;

process implementation process2.impl
   subcomponents
      t2: thread thread2.impl;
   connections
      cn: data port inport -> t2.inport;
end process2.impl;

thread thread1
   features
      outport: out data port;
end thread1;

thread implementation thread1.impl
end thread1.impl;

thread thread2
   features
      inport: in data port;
end thread2;

thread implementation thread2.impl
end thread2.impl;
```

*Figure 6. Connections description example*

The AADL language provides other possibilities to model embedded systems:

- **modes** : permit to describe the running modes of the architecture;
- **flows** : describe data and control flow , similar to the SysML *flow specification*;
- **packages** : permit to organize our model, this a principle present in SysML tool ;
- **annexes**: enable the use of declarations written in another sublanguage.

4. From SysML to AADL

The SysML model contains a lot of information about our system; it would be a loss of time and energy, to have to recreate that information to make an AADL model. However, certain detailed pieces of information such as the temporal properties of our system are often absent of a SysML model. In fact, SysML is usually used for high level design, and that type of temporal information is designed later in the process during detailed conception. Nevertheless, we could help reusing information contained in our SysML model not to waste time, and to ease com-

plete with the pieces of information the SysML model lacks. In this perspective, it is important to determine which are the possible links that can be made between those two languages.

SysML contain the architecture of our embedded system, and all the links between the components. Furthermore, the SysML BDD and IBD are close to the notion *type* and *implementation* in AADL. Indeed, the definition of *block* in the BDD permits to define the functional interface of our component which corresponds to the *type* of the component in AADL. The IBD of a *block* in SysML represent the same thing as the implementation of a component in AADL: the internal organisation of the component. Thereby, SysML's *parts* become components in AADL, *flow ports* become *ports* with AADL *type* depending on the kind of information they exchange, les *flow specifications* will be transformed in AADL's *flows*, …

The object-oriented approach of both languages allows a efficient translation of architectural concepts. Nevertheless, since AADL is a lower level representation, it uses more specific type of components. That's why to classify the components according to the 10 categories (Memory, processor, …) available in AADL, we have to consider another source of information to perform the model translation. The suitable answers are listed below:

- Impose a new way to model in SysML to make the AADL *type* present in the SysML model.
- Ask a specialist to classify each component. Using a questionnaire can make this step easier.
- Use a database of correspondences between SysML blocks and their category in AADL based on the record of the past projects.

A similar issue is observed to define all the properties that size the system, representing the quantitative properties needed to use properly tools as Cheddar [12] or RMA [13]. To completely define the *properties* of our system, we suggest the development of one of the three solutions presented before.

To manage those problems, we use a similar method than the one used to create FMEA and AltaRica DF models: using specialist's judgment to complete our model, and maintain a database of feedbacks for future projects. The use of the database will restrain the intervention of the experts as long as this database will be developed.

The steps used to create the AADL model are summarized below:

**Step n°1.** Identifying all the SysML blocks and parts and establishing the hierarchy between all those entities, taking into consideration the different levels of design.

**Step n°2.** Mapping every component with each others using ports and connections.

**Step n°3.** Categorizing each component of the system. (Ex: this block « shared_memory » belong to the category: *memory*).

**Step n°4.** Creating the structural model in AADL (textual and graphical model can be made at this point).

**Step n°5.** Filling in the properties that are not deducted from the SysML model.

**Step n°6.** Creating the final AADL model, which includes the structure description and the system properties.

| Step of SysML analysis \ AADL modelling steps | 1 : Components listing | 2 : Components classification | 3 : Proportion the system | 4 : Create the complete AADL model |
|---|---|---|---|---|
| Step 1 | ● | | | |
| Step 2 | ● | ● | ● | |
| Step 3 | | ● | | |
| Step 4 | | | | ● |
| Step 5 | | | ● | |
| Step 6 | | | | ● |
| DataBase | | ● | | |
| Specialist's judgement | | ● | ● | |

*Figure 7. Table of the steps of modelling and analysis*

It is visible that the steps 1,2,4,6 can be instantaneous with proper software, however steps 3 and 5 even using the database need to have recourse to a specialist, because some information may not be recorded in the database yet. The table from figure 7 sums up the correlations between the steps of the SysML model analysis and the steps which lead to the creation of the AADL model.

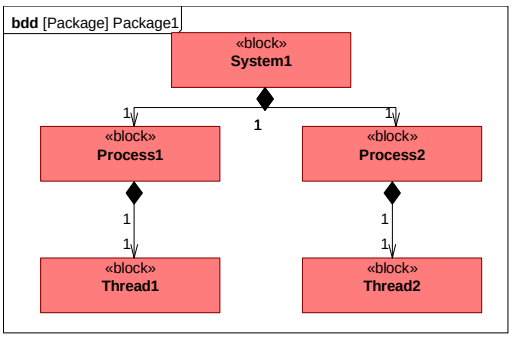


*Figure 8. Example of BDD*

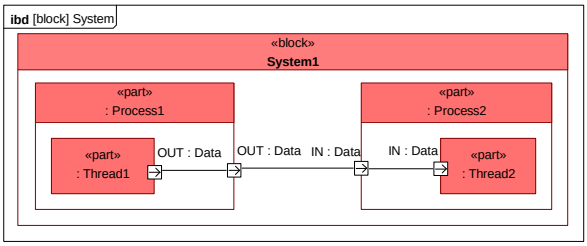

*Figure 9. Example of IBD*

For example, the SysML model corresponding to figure 8 and figure 9 permits to obtain the AADL model of figure 5 and figure 6 after realizing each of the steps defined before. Using the same method with our DMR, we can easily obtain an AADL DMR. In fact, dysfunctional behaviour is modelled with the same SysML artefact as the previous functional example.

If we compare the result with the extension released by the SAE in 2006 [14]: the error model annex which should provide tools to generate dependability studies, there is some difference. The main difference is the modelling of failure propagation: because our SysML DMR was made to ease FMEA analysis, the failure propagation is made through the fact that the data transmitted are corrupted and false, but no new signal is emitted, then a component must compute a diagnosis of their input data to detect a failure. It's very efficient to simulate the whole system in functional and dysfunctional mode and to study the real impact of a failure on the output data. But this method is too heavy to allow fault trees generation to study safety or Markov model generation to study reliability and availability.

Nevertheless it's necessary to specialize our DMR like showed in figure 10, as we did with to simplify the transformation to Altarica. The errors models of low level components for this type of studies capture the information needed for specific studies. Dependability analysis requires dependability-related information from the model: fault assumptions, repair assumptions, fault-tolerance mechanisms, stochastic parameters of the system (i.e., the occurrence of fault events and propagations).

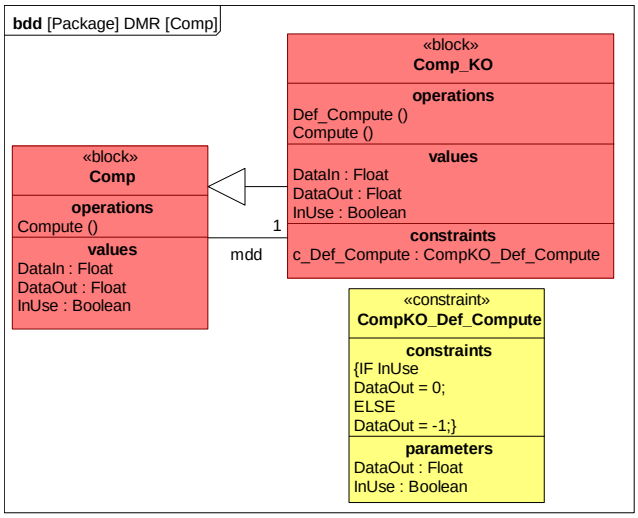


*Figure 10. Example of DMR in SysML*

Finally, the error model annex will permit to enhance the dysfunctional models of our components from the DMR in AADL. The tools provided by the error model annex are very useful to carry out dependability analysis of the system originally modeled in SysML which is used as the backbone of our entire method.

## IV. RESULTS AND PROJECT LEA LEARNINGS

Detailed results will be presented in the final paper. We will tackle the following points. We will explain how translation process from SysML to AADL allow us to obtained rapidly from specified architecture a timed model and proceed to a temporal analyze thanks Rate Monotonic Analyses [13] and cheddar tools [12].

We will focused on how memorize this information on SysML central model mainly with the help of parametric diagrams. At this stage we have the possibility to choose between several functional architectures. We will explain how we have made this choice guided by the following parameters: the Specifications of Physical ports, type of flows and their allowed requirements, bandwidth of candidate subsystems. Thanks the quality of FMEA process we have also took into account failure mode of the system to achieve the choice of the final architecture. We will present how FMEA process highlights the requirements impacted by dysfunctional mode and how it would direct our reasoning to choose the accurate functional architecture. In last, we will summarize the advantage to have two more axes of modeling: temporal and dysfunctional.

## REFERENCES


[1] P. David, V. Idasiak & F. Kratz. Improving Reliability Studies with SysML. *Proceedings of the 55th Annual Reliability and Maintainability Symposium, RAMS 2009, Fort Worth, Texas, USA*, Jan. 2009.

[2] P. David, V. Idasiak & F. Kratz. *Automating the synthesis of AltaRica Data-Flow models from SysML*. Proceedings of ESREL 2009, Prague, République Tchèque, 7-10 septembre 2009.

[3] S. Friedenthal, A. Moore & R. Steine. A Practical Guide to SysML : The Systems Modeling Language. *The MK/OMG press, Elsevier.*2008

[4] J. Estefan. Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev. B. *INCOSE MBSE Initiative, 23 Mai 2008.* 2008

[5] IEC 61508. International Electrotechnical Commission. Functional Safety of Electrical /Electronic /Programmable Electronic Safety-Related Systems. Parts 1 to 7. 1998-2005

[6] ISO 26262. International Organization for Standardization. Road Vehicles functional Safety. Standard under development.

[7] OMG Systems Modeling Language (OMG SysML) V1.1. *1st* November 2008.

[8] A. Rauzy. Mode Automata and their compilation into Fault tree. *Reliability Engineering and System Safety* 78: 1-12. 2002.

[9] Society of Automotive Engineers. *SAE Architecture Analysis &Design Language*. Specification V2, janvier 2009.

[10] C. Price, N. Taylor. *Automated multiple failure FMEA.* Reliability Engineering and System Safety Vol. 76, pp. 1-10, 2002.

[11] P. Teoh & K. Case. *Failure modes and effects analysis through knowledge modelling*. Journal of Materials Processing Technology 153-154, pp. 253-260, 2004.

[12] F. Singhoff, « The Cheddar AADL Property sets (Release 2.x) » LISyC technical report number singhoff-03-07, February 2007.

[13] Mark H. Klein, Thomas Ralya, Bill Pollak, Ray Obenza, Michael González Harbour, « A practitioner's Handbook for Real-Time Analysis », Kluwer Academic Publishers, 1993

[14] Society of Automotive Engineers. SAE Standards: AS5506/1, Architecture Analysis & Design