Model-Based Methodology for Requirements Traceability in Embedded Systems

Arnaud Albinet, Jean-Louis Boulanger, Hubert Dubois, Marie-Agnès Peraldi-Frati, Yves Sorel, Quang-Dao Van

Model-based Methodology for Requirements Traceability in Embedded Systems

A. Albinet[1], J-L. Boulanger[2], H. Dubois[3], M-A. Peraldi-Frati[4], Y. Sorel[5], and Q-D. Van[2]

[1] Siemens VDO, B.P. 1149, 31036 Toulouse, France. Arnaud.Albinet@siemens.com
[2] UTC/HEUDIASYC, UMR 6599, 60205 Compigne, France. boulange,vanquang@hds.utc.fr
[3] CEA, LIST, Boîte 94, Gif-sur-Yvette, F-91191, France. Hubert.Dubois@cea.fr
[4] I3S, UNSA, CNRS/INRIA, B.P. 121, Sophia Antipolis, F-06903, France. map@unice.fr
[5] INRIA, Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France. Yves.Sorel@inria.fr

Abstract. We present a model-based methodology for requirements traceability proposed in the framework of the MeMVaTEx project. The methodology relies on the EAST-ADL language and the two UML 2.0 profiles: MARTE for real-time embedded systems, and SysML for system requirements modeling. Along the paper, we illustrate the proposed methodology by an automotive case study, namely a knock controller, focusing on the time aspects of the requirements specified with the MARTE UML 2.0 profile. We explain how to define the requirements according to a proposed classification, and we present the tracing mechanisms based on the SysML UML 2.0 profile. Finally, we describe the proposed MeMVaTEx methodology which extends the EAST-ADL methodology in order to take into consideration the expression of requirements, and their traceability along the life cycle.
Keywords: end-to-end traceability methodology, application of traceability, UML model driven process, automotive domain, real-time, embedded.

1 Introduction

Embedded applications found in automotive domain continually increase in complexity not only according to the functionalities they must provide, but also according to the requirements they must meet due to multiple constraints such as dependability, timing, resources, variability, etc. For these reasons it becomes necessary to trace these requirements all along the development life cycle leading to the intended product in order to guarantee they are met. One of the major difficulties is due to the modifications that requirements must undergo from the highest functional level to the lowest implementation level. Indeed, they must be as consistent as possible to guarantee that traceability is efficient.

The paper presents preliminary results of a work achieved within the framework of the MeMVaTEx project[6]. This project is intended to provide a methodology for requirements traceability using a model driven engineering (MDE) approach in order to design

[6] This work has been performed in the context of the MeMVaTEx project (http://www.memvatex.org) of the System@tic Paris Region Cluster. This project is partially funded by the French Research Agency (ANR) in the “Réseau National des Technologies Logicielles” support.

embedded systems of the automobile domain. For this domain which demands a high level of dependability, sound methodologies are necessary to tackle the complex problems that arise. This project is closely related to other works carried out in the Competitiveness Cluster Ile-de-France System@tic, and in the European project ATESST, since they also concern MDE for embedded applications as well as the automotive domain.

Nowadays, dependability is a critical issue in automotive systems. The automotive industry is now conceiving its own standard[7] to introduce safety notions at every level of the development life cycle, from the system level down to the implementation level onto software and/or hardware. Since safety notions at each level are expressed by requirements, the control of safety-critical system depends on the control of requirements from their elicitation, their validation, and traceability through the different levels.

Requirement expression and traceability are key aspects of software engineering. The first studies on requirements engineering in the domain of software development has been started in 1990 [1]. Some tools like DOORS [2] handle the traceability management. More recently, the Paladin [3] approach proposed a requirement methodology based on UML in the domain of the web semantic. The SysML [4] UML profile allows now the designer to consider requirements as first class concepts in UML for system level design, and to deal with traceability concerns since relations between requirements and, requirements or model elements, are also defined in SysML. For the real-time embedded domain, the EAST-ADL [5, 6] language proposes a way to integrate requirements in the modeling approach process for automotive systems. Nonetheless, EAST-ADL neither covers all the requirement classes nor their traceability, and does not propose an integrated methodology.

Therefore, we propose a MDE methodology based on UML and its extensions (MARTE [7], EAST-ADL, SysML) for the modeling of requirements and their traceability in embedded systems. In this paper, we shall only focus on the three first levels defined in the methodology associated with the EAST-ADL language. However, it is planned that the five levels of the development life cycle will be covered at the end of the MeMVaTEx project. Consequently, the EAST-ADL methodology is extended in order to take into consideration the expression of requirements, possibly of different types, and their traceability along the development life cycle. An important issue concerns the transformation of the requirements when one proceeds to a next level because they request that corresponding rules are defined. We model the requirements and their traceability with the SysML UML profile that was defined for this purpose, while of course, relying on the EAST-ADL levels. Since timing constraint issues are of crucial importance for the dependability of the applications we are interested in, we use the MARTE UML profile to model accurately time relationships through the different levels of the development cycle. For other issues such as dependability, safety, availability, or security we merge the interesting features of SysML, EAST-ADL and MARTE.

We illustrate the proposed methodology by using an automotive case study, namely a knock controller.

The paper is structured as follows. Section 2 is an overview of the EAST-ADL language, and the EAST features we adopt in our methodology. Section 3 presents the knock control system focusing on the temporal characteristics. Section 4 focuses on the

[7] The IEC 26262, derived from IEC 61508.

requirement expression classification and modeling. The methodology is presented in Section 5. Some perspectives of this work are given in the conclusion.

2 The EAST-ADL process

EAST (Embedded electronic Architecture STudy)-ADL (Architecture Description Language) is a language for the development of vehicle embedded electronic systems. EAST-ADL was developed in the context of the EAST-EEA European project. It provides a unified notation and a common development process for all the actors of a car development (car-maker, suppliers,...). EAST-ADL allows a decomposition and a modeling of an electronics system through five abstraction levels (*Vehicle*, *Analysis*, *Design*, *Implementation*, and *Operational*). These levels and the corresponding model elements provide a separation of concerns.

The *Vehicle* level describes the main functionalities, and the variability points of the vehicle (Stakeholders view). The *Analysis* level models and refines these functionalities, and their interactions (Control/Command engineers view). The *Design* level represents a decomposition of functionalities with respect to allocation constraints, reuse, mode change, etc. (software engineers view). The *Implementation* level is an instantiation of the design level model. It produces a flat software structure which takes into account the software parts, the protocols and the OS (design engineers view). The *Operational* level consists in mapping the implementation model onto the effective ECU, frames, and tasks (design engineers view). An example of complete prototype car developed with EAST-ADL process can be found in [8].

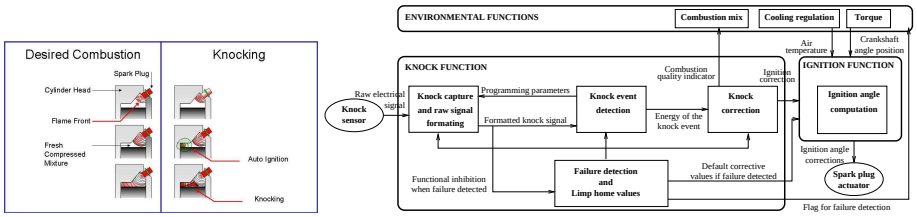
Another objective of EAST-ADL is to propose mechanisms to support variant handling, requirement expression, and requirements traceability. The requirements in EAST-ADL are an extension of those of SysML. They can be modeled either as a textual description or using a formal description, and they can be attached to create a dependence with any EAST-ADL objects. Requirements traceablility is also based on the same dependencies of SySML.

Today, EAST-ADL does not provide an integrated framework offering all these interesting aspects. Our methodology inherits from the EAST-ADL process, the different abstract levels (Vehicle, Analysis, Design, Implementation, Operational), but it enriches the EAST-ADL language with some model elements such has the expression of time, and the resources allocation. The traceability and requirements aspects follow a SysML syntax.

3 Case study: knock controller

We illustrate our methodology with the support of an automotive application: the detection and control of the knock phenomena. In gasoline internal combustion engines with spark ignition, an undesired effect may occur when the fuel mixture partially automatically ignites as a result of the compression in the combustion chamber.

Figure 1(a) shows the origin of the knock phenomenon. In a gasoline internal combustion engine, when the piston compresses the mixture, the spark plug produces a



(a) (b)

Fig. 1. Knock controller description.

flame front in the combustion chamber. The generated electric spark ignites the mixture, and produces uniform waves that push the piston to its original position. The very rapid heat release implied by this abnormal combustion generates shock waves that: decrease the engine performance, increase some pollutants that do not comply with norms, generate undesired engine vibrations, and decrease the user comfort. Eventually, it may cause a destruction of the engine, by damaging the spark or piston, potentially leading to jamming.

For theses reasons, monitoring the knock phenomenon is a critical issue. An appropriate anti-knock control, represented in Figure 1(b), is applied to each cylinder at every engine cycle from low engine speed up to the highest engine speed. A knock control system consists of one or several noise sensors, and a controller which acquire the noise through the sensors, and computes the correction during the combustion phases of the cylinders. Due to the speed of the combustion cycle in a motor, these treatments may be handled while satisfying complex real-time constraints during the design of the corresponding function. The knock control function equips a very wide range of engine management system for gasoline system. Several strategies for acquisition and correction are possible for optimizing the treatment of the knock, leading to manage a lot of variants for this function.

All these variants must be handled at the early stage of the design process. They lead to multiple solutions at the end of the design process. A first-part solution to manage this complexity consists in proposing a precise expression and classification of the requirements, using adapted models.

4 Expression and classification of requirements

Requirements are generally expressed by stakeholders or automatic control and software engineers with natural languages. In a design process, requirements must be validated and verified. On the one hand verification means that the designer must guarantee, at the different abstraction levels, that the requirement has been taken into account, and that they give raise to a corresponding model element. Verification is of particular importance in a certification process. On the other hand validation means that the designer

must guarantee, at the different abstraction levels, that the model satisfies the requirements.

We focus here on the verification of requirements. Since we develop a methodology based on the EAST-ADL process, requirements must be expressed at the different abstraction levels of this process. They must be refined, and a link must exist between requirements expressed at the different levels. A link between requirements is expressed using the traceability mechanisms of SysML. We propose a classification and a labeling of the requirements. These features permit the characterization of traceable paths for a requirement, or a class of requirements, through the levels of the EAST-ADL process. In the next section we present the different classes of requirements we have adopted. With the support of the knock example, we illustrate the SysML mechanisms provided for traceability, and show how to associate requirements to model elements of the methodology. For this purpose, we use the MARTE UML profile for addressing the expression of real-time requirement.

4.1 Definition and classification of requirement

The design of embedded real-time systems requires a precise management and tracing of user's requirements. It becomes even more critical when the design process must comply with a standard such as the IEC 880 in nuclear system, CENELEC EN 50126, EN 50128 and EN 50129 in railway system in Europe, and DO-178 and DO-254 in aeronautic system. Above all these standards there is the general IEC 61508 standard [9] which concerns all systems based on electric, electronics, and programmable electronics. These standards recommend the application of requirement engineering with end-to-end traceability applied to the whole V-cycle. Details and discussions about these standards can be found in [10].

Requirements are generally expressed in natural language. Some research initiatives are provided to transform the expression of needs into a set of requirements. A simple description of a requirement is not fully sufficient to define it. There is other status information that each requirement must carry. The requirements must be tagged to provide such information.

For that purpose, we consider a requirement as a structure with several attributes. A requirement is characterized by an *identification*, a *textual* description, and a *type* (*functional*, *non-functional*). The non-functional type is refined into sub-types such as reliability, performance, safety, and cost, etc. Some other attributes indicate the *derivation type* (decomposition/refinement), the document source where the origin of the requirement can be found, the *verification method* that must be applied on this requirement, and its *agreement status*. Some of these attributes are automatically generated. For example, the identification attribute consists of a number, and a label representing the level in the EAST-ADL process (Vehicle Level VL, Analysis Level AL, Design Level DL, etc.) Other attributes are defined by the user (Functional/Non-Functional), others attributes are flags which are set after an analysis phase (Agreement status).

Table 1 gives some examples for the identification and definition of requirements for the knock controller.

This precise labeling of requirements becomes essential when a traceability process is requested. It is important to demonstrate that all input requirements are satisfied at

Table 1. Example of requirement expression.

Req. Name	ID	Text
Eng.Pha.Pos.	VL-F-2	The engine management must detect the different positions of the cylinder and control the ignition in an optimal manner.
Eng.Cam.Pos.	AL-F-4	Observing the camshaft position, the knock function must locate the ignition phase in a 4 stroke cycle.
Eng.Cra.Pos.	AL-F-8	Observing the crankshaft position the knock function must detect which cylinder is concerned by the knock correction.
Kno.Con.Dur.	AL-NF-P-2	The filtering and the detection/correction must be executed before the next ignition phase of the same cylinder.
Acq.Dur.Con.	AL-NF-P-3	The acquisition duration window must be large enough to acquire at most 2 samples of the knock sensor.
Rot.Spe.Val.	AL-NF-D-2	The rotation speed for the motor is lower than a constant MAXRPM.

the lower levels (i.e. they are linked to other requirements, or they have been taken into account by a model). This demonstration is based on links established between requirements, and on arguments associated to these links.

We use SysML to express the requirements and their relationships. The section 4.2 presents the different constructions taken from SysML for this purpose.

4.2 SysML requirement and tracing mechanisms

The SysML profile (for Systems Modeling Language) is a specialization of UML for systems engineering applications. SysML supports specification, analysis, design, verification, and validation of various systems potentially complex. SysML is defined as an UML profile since it uses a subset of UML 2.1 [11]. SysML extends UML with new notations and diagrams such as the requirement, and parametric diagrams which are an extension of internal block diagrams. We focus on requirement diagram since this is the most appropriated for our considerations.

The purpose of requirements in SysML is to provide a relationship between requirements, as traditionally managed, and model elements as usually managed in UML. Thus, a requirement is a stereotype of a UML class that is subject to a set of constraints. A requirement in SysML is composed of an identifier and a text that describes the requirement in a natural language. Additional properties can be attached to these two fields. We use this functionality in our approach.

Several links are defined in SysML to express requirements relationships, and to link them to other model elements. The different relationships are the requirements hierarchy *(refine)* to describe how a model element can be used to further refine a requirement, the derivation *(derive)* that corresponds to requirements at the next level of a hierarchy, the requirement satisfaction *(satisfy)* which describes how a design or an implementation model satisfies a requirement, and the verification *(verify)* which defines how a test case verifies a requirement. With such relationships, initial requirements can be traced by following a top down path in the requirement tree. In reverse, a requirement at any level can be traced back to its origin. Supported by these previous SysML relations,

traceability between requirements and model elements, are considered directly in the models; such as traceability between requirement evolutions in the model development. The Figure 2 illustrates a requirement diagram and relationship between requirements and model elements for the knock controller case study.

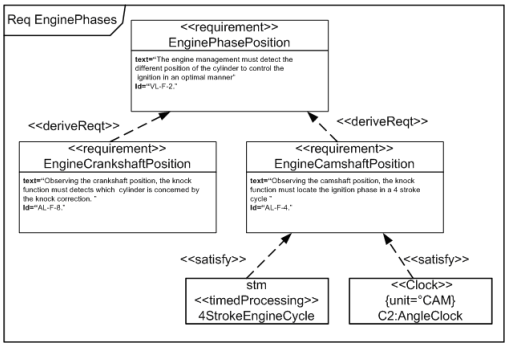


Fig. 2. Example of a SysML requirement diagram.

4.3 MARTE profile for real-time requirements modeling

Real-time constraints belong to the non-functional requirements class. The MARTE profile is used in conjunction with SysML to model the temporal and allocation characteristics of an embedded system. MARTE extends the modeling capabilities of UML and SysML, and makes clear the semantic with an objective of model validation.

In this section we pay a special attention to the time model of MARTE which allows the modeling of multi-clock systems. As embedded systems interact with mechanical and physical components the software computing part is usually triggered by heterogeneous events. A clock can be either associated with the classical time (second, hertz) or a logical one (round per minute, angular position, meter).

The knock control example is a good candidate for multi-clock system modeling. The capture phase depends on the period, in hertz, of the acquisition sensor whereas the filtering and correction phases are triggered by events the occurrences of which are measured in angle degrees. This possibility to deal with such logical time is adapted to the specification expression of embedded systems. It allows a manipulation of time at a high level of abstraction, and a direct relationship between the requirements and the model. The definition of clocks in MARTE corresponds to the definition of a class named AngleClock which is the time base that represents the temporal evolution of a rotation. From this class we declare two instances of this clock, camClk and crlClk . These clocks represent the revolution position of the camshaft and the crankshaft which are crucial mechanical elements of an engine. The definition of these two clocks participates to the satisfaction of the AL-F-4 and AL-F-8 requirements (cf. Figure 2). These two clocks have a resolution, an offset, and a maximal value (modulo).

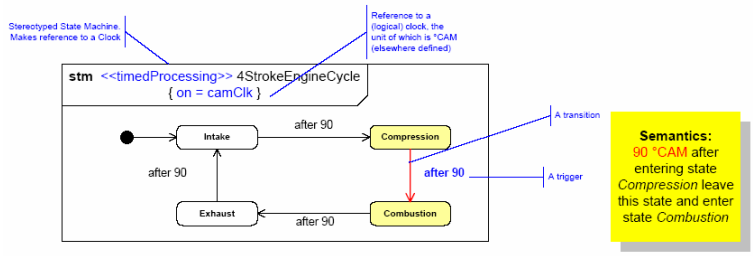


Fig. 3. State machine of a four stroke cycle.

These clocks can trigger some processing as it is showed in Figure 3. The timedProcessing elements are UML behaviors that can be triggered by logical events or clocks. Some duration constraints may be expressed on these behaviors. For example, the requirement AL-NF-P-2, which belongs to the performance class, is satisfied by a duration that must be associated as a constraint to the knock controller block. ≪timedDurationConstraint≫{KFilter.Duration+KDetect.Duration<720°CAM}

This capability to express at the same level of abstraction the requirements and the model is a way to bridging the gap between the requirements and the model, but also to link the requirements with the underlying properties to be verified.

5 MeMVaTEx methodology

We propose a methodology that considers requirements as a first class concept, from the early steps of modeling, and during all the phases of the development life cycle. The MeMVaTEx methodology aims at merging different languages and standards: UML2.0 and SysML are used for system modeling and requirements traceability in the models, the EAST-ADL language is used for the automotive architectural description. The MARTE UML profile deals with the real-time constraints modeling. Furthermore, the EAST-ADL process is adopted as a standard to provide the different modeling abstraction levels (see section 2).

As shows in Figure 4, the proposed methodology keeps the five levels of the EAST-ADL process for structuring the development. At this stage of the project we have only addressed the vehicle, analysis, and design levels. For each level, two branches evolve in parallel. The requirement branch on the left side allows expressing, defining, and tracing the requirements of the system. Requirements are expressed in such a way that they can be managed by dedicated tools; the modeling branch on the right side is composed of models integrating related requirements. A set of heterogeneous models can be explored at each level for expressing different parts of the system (behavior, architecture, algorithms, allocations,...) For instance, UML/SysML models are built with the ARTiSAN Studio tool, and the behavioral functions with the Simulink tool. In order to manage the requirements we use Reqtify, a light and powerful tool who facilitates the traceability: textual requirements – stored in Word or Excel documents – are imported to Reqtify [12], and are represented in our model. Each requirement is linked to another

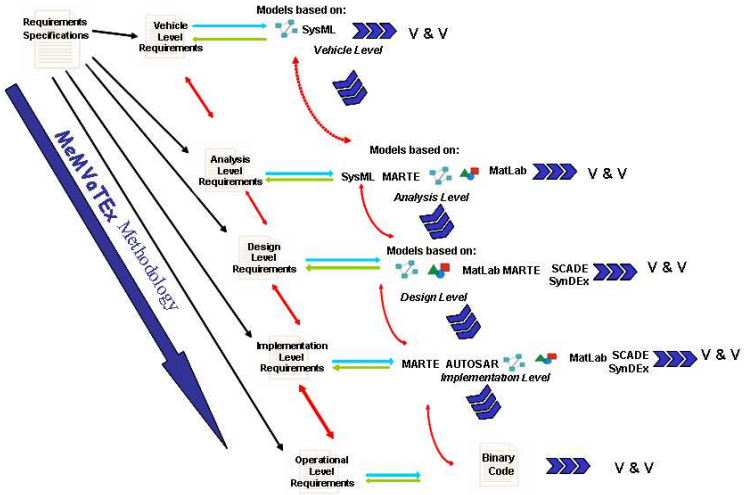


Fig. 4. The MeMVaTEx methodology.

requirement, or a part of the model, or a test case. Reqtify manages these links and can export analysis and traceability reports at any level during the development process.

The general outline presented in Figure 4 shows that initial requirements are given through a textual form, and must be dispatched at the different levels. The originality lies in the fact that these requirements are considered along the whole application development life cycle, and built at each level. Additionally, some requirements can appears at intermediate levels, like exported requirements, and for these reasons, it can be confusing to trace back these requirements to formers levels. Requirements are traced among the modeling branch by using the traceability mechanisms offered by the SysML metamodel (composition, verification, satisfiability, etc.) This traceability can also be followed between the corresponding requirements in the requirement documents.

From this point, requirements models are defined, and traceability is achieved between requirements pools and models. This is done by building references between requirements elements included in the models and requirements in external documents. These external documents (on the left branch in the Figure 4) are requirements pools. In the project, the references between the initial requirements documents and the requirements model elements are managed by traceability tools like Reqtify or its open-source release in Topcased, TRAMWAY.

Model elements used at each level are carefully selected in the methodology. We have voluntarily limited the use of some models because they are not endowed with a sufficiently precise semantics giving raise to ambiguous models. All these choices are made with the objective to connect the models with validation and verification (V&V) tools. These tools will check that models satisfy properties induced by requirement expressions. The connection with V&V tools and technologies will be facilitated by the MARTE profile that includes the analysis part of UML models for real-time systems; it will also be facilitated by the use of Matlab models and its connection with the Simulink

tool for the design levels. The MeMVaTEx methodology achieves a major goal for industry in the domain of system and software engineering based on model driven architecture. It allows the designer to facilitate the use of common tools, an easier update and evolution of the models, and the requirement integration along the MeMVaTEx process.

6 Conclusion

We presented the MeMVaTEx methodology for requirements traceability through the first three levels of the EAST-ADL process in the case of a knock controller, example coming from the automotive domain. Relying on the two UML 2.0 profiles: MARTE for real-time embedded systems, and SysML for system requirements, it allows the designer to express the requirements according to a proposed classification, and to trace them along the EAST-ADL process.

This method will provide a competitive advantage to the industry, a mastering of the system development quality, and will reduce the cost of re-engineering by decomposition of solution, and backward impact analysis (reuse) centered on the requirement management.

As future works we plan to extend the methodology to the five levels of the EAST-ADL process taking into account the requirement related to the hardware architecture, the operating system, and the allocation of functions to the hardware components.

References

1. Thayer, R.H., Dofman, M.: System and Software Requirements Engineering. IEEE Computer Society Press Tutorial, 1990.
2. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer, Second Edition, 1995.
3. Mayank, V., Kositsyna, N., Austin M.A.: Requirements Engineering and the Semantic Web. Part II. Representation, Management and Validation of Requirements and System-Level Architectures. ISR Technical Report 2004-14, Univ. of Maryland, College Park, 2004.
4. Object Management Group. The Systems Modeling Language, 2006. http://www.sysml.org
5. ITEA Project Version. EAST-ADL: The EAST-EEA Architecture Description Language. 2004. http://www.east-eea.net
6. Debruyne, V., Simonot, F. and Trinquet, Y.: EAST-ADL an architecture description language – validation and verification aspects. IFIP, WADL04, 2004.
7. Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). Request For Proposals, 2006.
8. Lönn, H., Saxena, T., Nolin, M., Törngren, M.: FAR EAST: Modeling an Automotive Software Architecture Using the EAST ADL. ICSE Workshop on Software Engineering for Automotive Systems, Edingbourg, 2004.
9. International Electrotechnical Commission. Functional safety of electrical/electronic/programmable electronic safety-related systems. Std. 61508, 2000.
10. Boulanger, J-L.: Expresion and Validation of Logical and Physical Safety Properties for Critical Systems, PhD thesis (in French), Université de Technologie de Compiègne, 2006.
11. Object Management Group. Unified Modeling Language: Infrastructure, version 2.0, Document formal, 2006.
12. Larronde, E., Burgaud, L., Fourgeau E,: Shift towards a Cohesive Design Based Management of Automotive Embedded Systems Requirements. European Congress on Embedded Real Time Software, 2006.