

Requirement Analysis and Development using MATLAB Models

Jinming Yang, Jason Bauman and Al Beydoun

Electrical Systems Division, Lear Corporation

Copyright © 2009 SAE International

ABSTRACT

Requirements development and analysis for automotive electronics products have been found to be tremendously challenging to both OEMs and suppliers. Besides ambiguity, incompleteness, conflicts and other pitfalls commonly seen in requirement specifications, in some cases, a requirements document for a new electronics product may even not exist and may need to be developed from scratch. Analysis reveals that generic model-based approaches and toolsets presently available lack support for requirements development while facilitating all other development activities across the entire product development cycle.

In this work, we describe a model-based development methodology centering on requirements development, engineering, and management while supporting other development activities including requirements analysis and clarification, rapid prototyping, simulation, verification and validation, automatic code generation, and SIL/HIL testing. The methodology leverages the MATLAB® toolset features of visual conceptualization, rapid prototyping, simulation and ease of accessing model property data with m-scripts, not only to improve the traditional requirements development activities for understanding customer's needs, abstraction and definition from concepts, requirements document elaboration, but also to make it possible to maintain a set of executable models that can be readily simulated and verified throughout the entire development process. We demonstrate that the methodology shall be particularly suitable to developing requirements from scratch and/or evolving a high-level requirements document to detail-level requirements specifications.

INTRODUCTION

It is well-recognized that many failures and deficiencies of automotive electronics products are primarily due to incomplete, ambiguous, conflicting, unrealistic, and/or untestable requirements specifications. In general, such requirements problems can be attributed to the lack of a commonized platform and adequate joint effort exerted in the early requirement development stages between requirements and design teams. One reason is that requirements writers and development engineers often have divergent goals and different perspectives. Typically, a requirement expert may not know implementation details while a design engineer may lack system overview (the "big picture") for an overall system. The situation was aggravated even more by ever-increasing requirements complexity due to the trend of more and more functions pushed to be implemented in software as program memory spaces and CPU horsepower on a microcontroller have been drastically increased in the past decade. Consequently, problematic requirements specification documents combined with more complex functions are jointly driving up embedded software cost increases and, more importantly, crippling productivity and competitiveness. This is why companies supplying automotive ECUs are often struggling to deliver complex, high-quality software products and they often end up shipping products late, delivering less than acceptable software and running over budget [1].

This paper is organized as follows. We first analyze legitimate issues existing in requirement documents for automotive electronics products and why a model-based requirement development methodology is necessary. We then describe our methodology and summarize the

benefits of the model-based requirement development methodology described in this work. A case study for creating a requirements document for a rear wiper controller using MATLAB models is presented. With the aid of the case study example, we describe the key points in our model-based requirement development process.

REQUIREMENTS ISSUES

Occasionally, a formal requirements document may need to be developed from scratch for a new electronics product. In this case, the requirements would be in forms of meeting minutes, hand drawings, drafted notes, e-mails, documentation from similar products, or combinations of the above. In this work, we treat such requirements as problematic requirements documents, for which our requirements development and engineering methodology is suitable and targeted as well. Generally speaking, requirements documents used for developing automotive electronics ECUs have various issues and/or problems, which are elaborated and analyzed as follows.

MIXED WITH IMPLEMENTATION DETAILS

While ideally only external system behavior (*i.e.*, black box) should be specified, a requirements document may be mixed with implementation details. Because of the complexity of automotive embedded software, both textual requirements and a variety of forms of representations (*e.g.*, charts, diagrams, tables, *etc.*) have been employed in practice [2]. As a result, the distinction between requirements and implementation specifications tends to get blurred by mixed information of functional requirements and implementation details. While some of the implementation specifications may be “literally” adopted in implementation, a majority of requirements representations have to be “tweaked” or modified before they make sense to design engineers. This process will more or less make implementation deviate from the intended functions which will then demand additional design reviews and requirements changes. In this case, requirements specifications become unstable which can impact the project plan and timing schedule.

CONFLICTING REQUIREMENTS

When more and more functions are crammed into one single silicon chip, the system being developed may have to be divided into smaller pieces in forms of sub-systems, functions, sub-functions, and components. As more manpower is teamed up to create requirements for a large system, engineers from different fields will have to work in parallel in order to meet the project schedule. In this case, the bottom-up and top-down requirement development strategies shall inevitably be both employed. Consequently, persons who go through the bottom-up path may lose the “big picture” for the system,

while those who create the requirements from top-down may lack the detailed knowledge for the system that can be classified as multi-stakeholder distributed system as described in [3]. This is one of the reasons why quite often, many conflicting specifications come into existence in requirements for a large system.

PARALLEL PROGRAMMING PITFALLS

When various sub-systems, functions, sub-functions and components are specified and designed in parallel, parallel programming methods are *de facto* adopted (not even taking into account of a multi-core processor). Therefore, it will be natural for all the sub-systems and sub-functions to be implemented in parallel and thus to be concurrently executed in the intended target system. As a result, it will be very likely that the problems in parallel programming such as race condition, deadlock and starvation [4] shall come into the scene across various sub-systems and sub-functions. Software bugs associated with such parallel programming issues are usually very hard to find [5] and most of them can hardly be identified until later in the development phases. Plus, resolving such issues may deviate from customers’ intention. In this case, design engineers need to communicate with the customer using certain simulation tools.

ISSUES FOUND IN VALIDATION

From our past experience, many requirements issues can manifest themselves not only in the earlier product planning phase but also during the later phases of design, implementation and system validation. Due to the fact that a requirements document may be developed by multiple engineers from various fields, some of requirement issues and problems can only be identified and resolved in the later development phases when sub-systems and/or functional blocks are integrated. For certain products using new technologies, a complete requirements document simply does not exist after kicking off design and implementation activities. In this case, the requirements document needs to be developed, revised and evolved as the product is designed and implemented in an iterative manner.

REQUIREMENTS FOR EXCEPTIONAL PATHS

Many requirements only specify acceptable responses (*i.e.*, the “happy paths”) but do not define undesired events (*i.e.*, the exceptional cases). Often, a system that produces outputs specified in the requirements may also yield unexpected and/or unacceptable responses when certain combinations of stimuli input vectors are applied to the system. In this case, an executable models that simulate the system behaviors (what should do as well as what should not do) will be essential.

A MODEL-BASED METHODOLOGY FOR REQUIREMENTS DEVELOPMENT

For the aforementioned reasons, an effective toolset that can be used to resolve or at minimum, to alleviate the requirements issues becomes important. The model-based development methodology described in this work gives emphasis on requirements development and engineering while not ruling out other model-based design activities including requirements analysis, test cases-based requirement clarification, rapid prototyping, simulation, verification and validation, automatic code generation, and SIL/HIL testing. The ideas behind this methodology are summarized as follows.

- (1) Requirements models that reflect and simulate desirable system behaviors are created in accordance with the hierarchical structure of the given requirements document. As MATLAB models support both top-down and bottom-up approaches, requirements models can be developed from both directions.
- (2) Layerization and modularity of the requirements models are realized using the *Subsystems* blocks in Simulink® and the *Sub-charts* in Stateflow®. This practice helps manage system complexity as well as facilitates requirement changes and revisions.
- (3) An executable virtual test harness model is created together with system-level test vectors and test cases by which system behaviors are simulated, visualized and investigated. Requirements issues are then identified, root-caused and resolved in the earlier stages of product development.
- (4) Using the test harness model, test cases for the purpose of requirement verification and clarifications are created. Due to the heuristic nature of model-based methods [6], any requirements issues can be scrutinized by creating additional test cases that magnify and stress-test dubious spots (e.g., statements, timing charts, etc.). All these test cases can be retained and evolved for both SIL and HIL testing.
- (5) The MATLAB® programs (m-scripts) provide an effective means of accessing model properties such as textual descriptions of the Simulink blocks and variables used in Stateflow charts, variables names, data types, scopes, etc. There are quite a few benefits by using custom m-script utilities, which include, (a) a requirements document can be generated from the requirements models, (b) a data dictionary can be generated and maintained that is usually crucial to developing a large system.

- (6) An executable requirement model can be a good start point for evolving it to a behavior and/or an implementation model which can be used for code generation. Thus, problems and issues identified in various development phases can be propagated either down or up so that inconsistencies are eliminated or minimized.

CASE STUDY: REQUIREMENTS MODELS FOR A REAR WIPER CONTROLLER

The following illustration uses the development of a requirements model for a rear wiper controller as an example. The rear wiper control may be a functional block in a body-control system or a standalone ECU module. The central requirement for the wiper control is to sample the status of the rear wiper switch and a rain sensor. In modern vehicles, the rear wiper may also be controlled by the reverse gear and the front wiper switch, that is, when a vehicle is put in the reverse gear while the front wiper switch is on, the rear wiper wipes. More advanced vehicles may require the wiper speed to vary with the vehicle speed. We assume that the wiper functions are incrementally added on during the development phases. Particularly, the ten requirements items are divided into three groups and listed separately in the three sub-sections below. The first group of requirements represents the basic functions for the rear wiper controller, the second group represents the added requirements that make an advanced controller, and the third group are the requirements that make a full-blown rear wiper controller. A MATLAB model will be evolved to reflect the requirement changes and improvements.

REQUIREMENTS MODELS FOR REAR WIPER CONTROLLER

Requirements for basic functions

Now assume that the basic requirements for a simple rear wiper controller are:

1. *The rear wiper shall be turned off if the ignition is not in the "RUN" position, regardless other conditions.*
2. *The precondition for the wiper control being turned ON is when the ignition is in RUN position.*
3. *The rear wiper control is turned ON when the rear wiper switch is turned on and when Ignition==RUN.*

The requirement model is designed as shown in Fig. 1.