MODEL-BASED SYSTEMS ENGINEERING OF AUTOMOTIVE SYSTEMS

Gerard H. Fisher, Vitech Corporation, Vienna, VA

Abstract

For as long as systems engineers have been around, people have thought of them as "paper pushers." The primary products of systems engineers all seem to be documents such, as specifications and trade-off studies. Communicating the requirements to hardware and software designers with documents is fraught with problems. By using a model-based approach to systems engineering, there is no ambiguity. The model fully defines the functional behavior, inputs and outputs, and the physical architecture, as well as the performance and resource requirements. This method provides a unified, consistent and traceable design.

This paper describes the design of the Automotive Personal Assistance System (APAS) exploiting the features of the Global Positioning System, which is similar to the Cadillac OnStar System. It includes the entire systems engineering life cycle, from source requirements analysis, through behavior analysis and physical architecture, to verification and validation. The behavior model can be fully executed to verify its correctness long before money is wasted building prototypes. The use of the model-based approach will aid in the reduction of product cost and the increase in product quality.

Introduction

Systems engineers (SE) have been prolific producers of proscriptive prose. The SE's most prominent products are documents; the heavier, the better. Systems engineers perform many tasks, but always convey their results in the form of a document (e.g., system specification, unit specification, trade-off study report, etc.). When the SE passes a specification to the hardware or software engineers, there is a tremendous opportunity for miscommunication. Because of the multiple meanings of English words, it is easy to misunderstand the author's intent. Furthermore, words are not the only requirements. Requirements may also come in the form of relationships, attributes and diagrams. It is very difficult to perfectly coordinate all of these items. Using an object-oriented model-based tool, such as CORE®, for system development will greatly aid the systems engineer in this task.

This paper discusses the mythical design of a automotive system to aid the driver through the use of Global Positioning System (GPS) data. Cadillac offers such a system called OnStar. There are also several "after market" versions available.

Originating Requirements

Figure 1 illustrates the simple requirements for the APAS in the form of a context diagram. As the requirements are analyzed, it is useful to draw such a diagram to help understand the overall context for the system to be designed. The large ellipse represents the system of interest. The smaller ellipses show the external interfaces to our system. Key inputs and outputs are shown as labeled arrows.

The APAS is required to perform the following:

- Receive and process Global Positioning System data.
- Alert monitoring station of automobile accident

- Remotely unlock doors at owner's request
- Monitor & assess its own performance
- Retain automobile position history for 24 hours.

In this example, the customer came back later with some additional requirements (ECP #1), which added the following:

- Alert monitoring station of auto theft and current position
- Assist owner in finding car in parking lot by sounding horn and flashing lights.

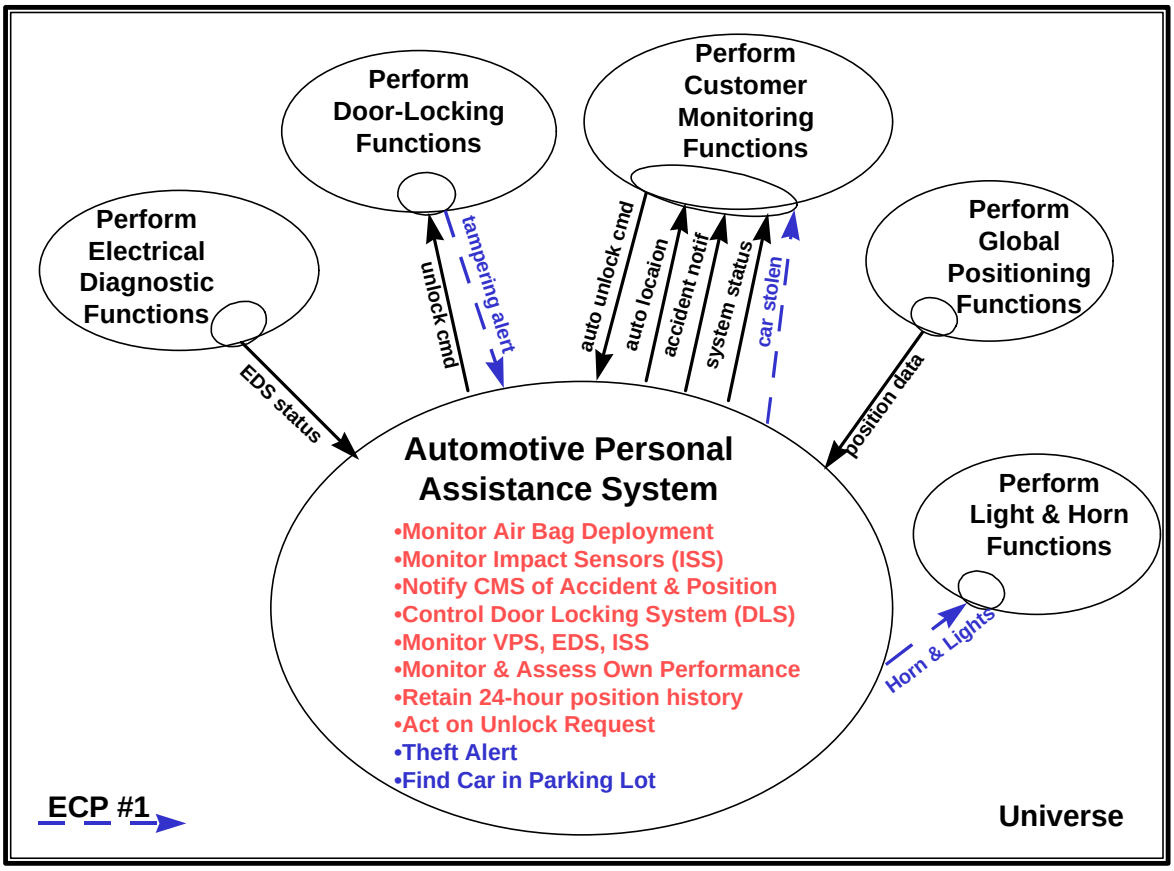

Figure 1. APAS Context Diagram.

These requirements are analyzed by the systems engineer and extracted. Compound requirements are broken into simple statements. Surely some of the requirements will be ambiguous. Those ambiguities are usually handled by creating an action item or *issue*.

The issue describes the problem associated with the originating requirement and assigns a severity, ownership and a due date. These ambiguities are perhaps the major impediment to progress in the initial design phase. Customer TBDs (to be determined) and ambiguous requirements will stretch the schedule and add cost to a project. Once sufficient information is obtained, the issue should document the assumptions made, the alternatives considered and the eventual decision, along with the usual configuration control data. By keeping the customer informed of these issues and diligently tracking them to conclusion, one can minimize the impact. Figure 2 illustrates the requirements hierarchy for the basic re-

quirements for APAS. It traces the requirements from their source specification through high-level compound requirements to the leaf-level, implementable and verifiable requirements.

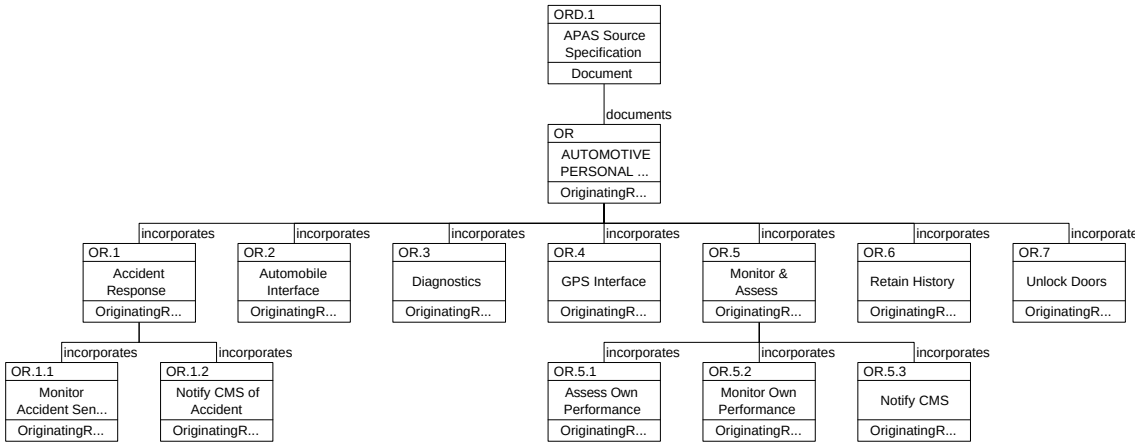

Figure 2. Requirements Hierarchy.

Behavior Modeling

Once all requirements have been extracted, analyzed and understood, it is time to postulate one or more functional models that would fulfill these requirements. All requirements should trace to a function. If you end up with functions that do not trace to a requirement, the function may be superfluous. If, on the other hand, you find requirements without a corresponding function, your model is incomplete. The models should be constructed in layers. The top layer should be similar to the context diagram described above.

An example of such a high-level model is shown in Figure 3, in the form of an enhanced function flow block diagram (E-FFBD). The system under design is "Operate APAS." The other functions shown represent the external interfaces to APAS. These include the GPS interface, the door lock system in the car, the automobile's electronic diagnostic system (EDS), automobile's horn & light systems, and the customer monitoring station (CMS) that supports calls for help. The *and* construct indicates that all functions are operating in parallel. The functions have two types of inputs and outputs shown. Those with only a source or a destination are called *data stores*. The data is assumed to be available whenever it is needed. The other type of data element is called a *data trigger*. A data trigger is generated by one function and sent to one or more other functions. The functions requiring the data trigger cannot begin until it arrives. This feature permits the E-FFBD to be simulated.

The use of a discrete event simulator is very helpful in verifying the logical correctness of the design. Whenever possible, a systems engineer should postulate two or more behavior models and trade them off to select the best design. The simulator is one of the tools that will aid in these trade-offs. However, this type of trade-off requires the combination of economic, performance and subjective data to arrive a single figure of merit. Another technique that would be useful for this task is the analytic hierarchy process (AHP).

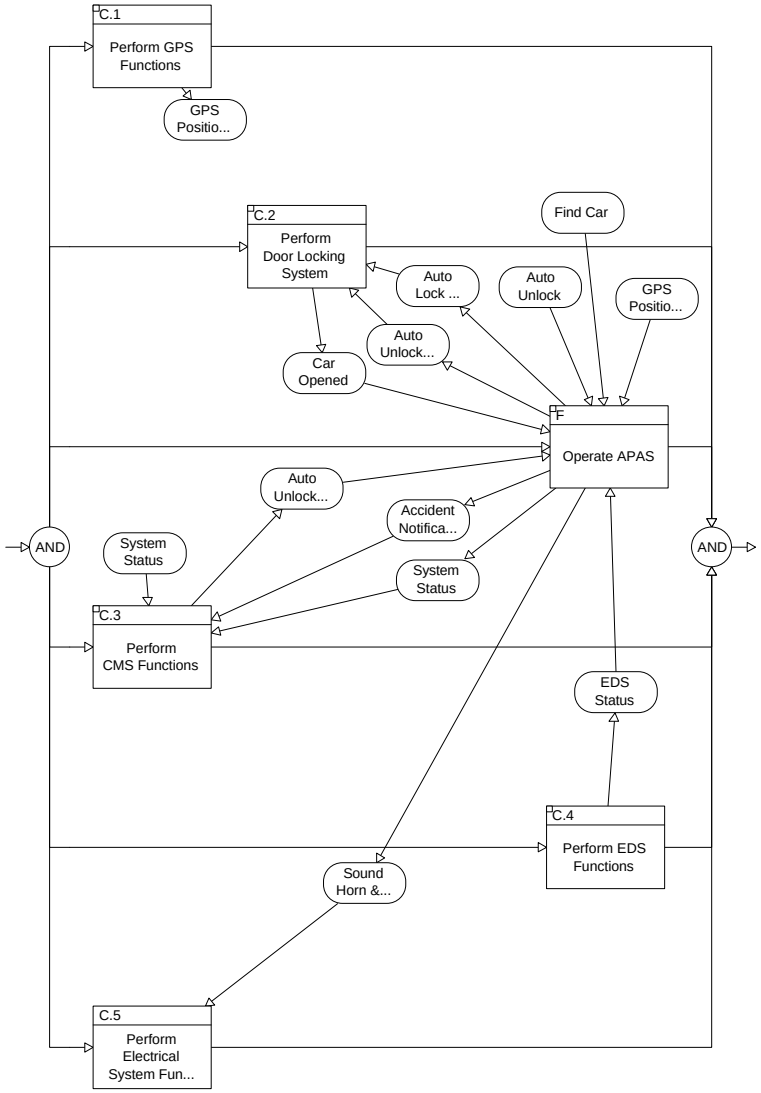

Figure 3. Functional Context Diagram.

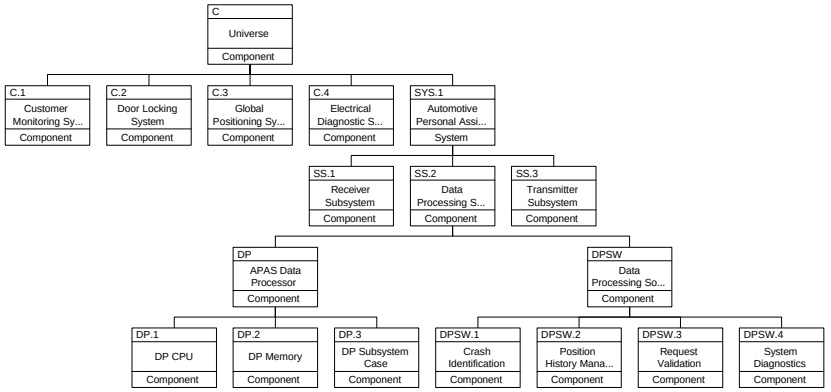

Figure 4. Physical Design Hierarchy.

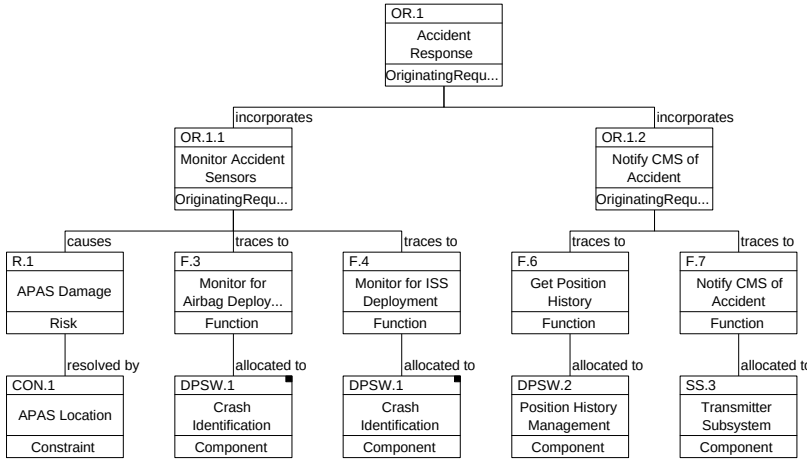

Figure 5. Traceability from Requirements to Components.

AHP is a form of multi-attribute decision analysis, developed by Dr. Thomas Saaty in the early 1970's. AHP includes development of a hierarchical model of the criteria to be used in a decision and then develops a set of relative weights using a pairwise comparison method. The SE might make a series of Monte Carlo runs for each proposed behavior model, measuring task times, wait times, queue sizes, etc. These results, along with other measures and estimates, could be plugged into an AHP model to determine the most compliant design.

Physical Design

Once a preferred behavior is selected, the SE can begin developing several physical designs. As part of the physical design, the systems engineer will allocate functions to hardware, software, and humans. These are the three types of physical components in a system. The way in which these three function together is the *system* or *process*. In a similar fashion to the process above, these designs should be traded off to select the preferred design. If one were to give the same requirements and behavior model to three experienced systems engineers, the results would be three unique system designs. There are always some obvious choices for software implementation or hardware implementation, but some require serious trade-offs. This could be another application for the AHP. Some of the key parameters one considers in physical design trade-offs are cost (both initial and recurring), power consumption, performance, reliability, maintainability, supportability, technical risk, etc. Figure 4 illustrates the physical design hierarchy for APAS. The interfacing systems are numbered C.1 through C.4. Our system, SYS.1, has three major subsystems: receiver, transmitter and data processing. The data processing subsystem is comprised of both hardware (the data processor) and software (the data processing software). Each of these can then be broken down to as low a level as necessary. However, the systems engineers should cease design when they have allocated function to the specialty (HW or SW). Let the specialty engineer make the design and implementation choices below that level. This is not to imply that the specialty engineers should use a different method. They can and should stay with the same model-based approach throughout their design phase. When the hardware engineers are ready to design VHDL (the hardware language used to specify design) and the software engineers are ready to code, they can just export the design from the tool. There are several tools that interface with VHDL and code generators. As a final check of your design, Figure 5 shows you a trace from specific requirements through functions to physical components. The inverse would ensure that all physical components had a reason for being in the design. If they could not be traced back to a specific requirement, the component might be unnecessary.

Verification and Validation

During the requirements analysis phase, a key step is the assurance that compliance with a requirement can be proven. If not, the requirement is not valid and the issue should be resolved with your customer. If the requirement can be verified, then a high-level *verification requirement* should be generated during the requirements analysis phase. You have though it out and it is a good idea to record it while it is fresh in your mind. For instance, if a requirement specified a 95% isolation of all hardware faults to a single circuit board, you might specify the verification requirement "verify 95% fault isolation to a single circuit board by doing 100% fault insertion on all input / output pins on the board." This would give you enough information to be able to estimate the cost of the effort. When detailed test planning is done, that verification requirement would be expanded to describe a *verification event*, assign a responsible individual, develop a test plan and procedure, and identify the test configuration needed to perform such a test.

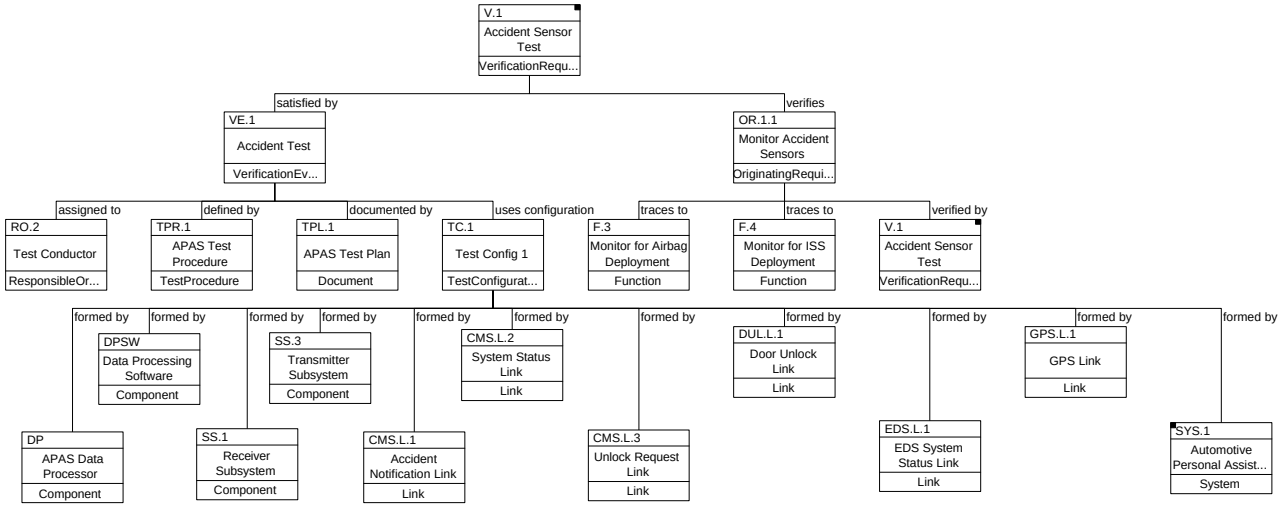

Figure 6. Verification Hierarchy.

Figure 6 illustrates the verification hierarchy for the APAS Accident Sensor Test. In addition to the information described above, it also identifies the requirements and functions to be verified. This becomes especially useful in large systems, where test resources are at a premium. Being able to identify the equipment needed will permit scheduling of tests to avoid conflicts.

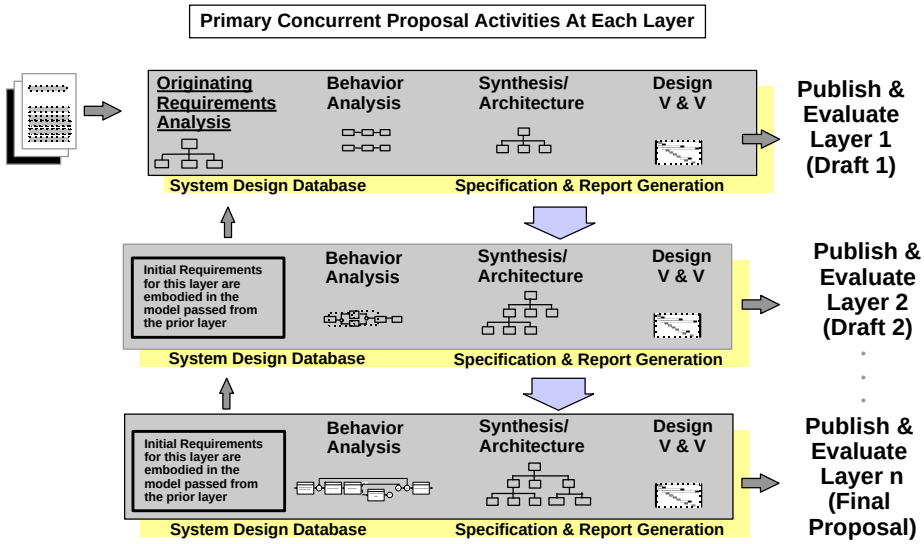

Figure 7. The Onion Model.

The Onion Model

Figure 7 illustrates the recommended method for doing model-based design. First extract high-level requirements and develop a high-level model for behavior. Then develop a corresponding high-level physical design and verification and validation (V&V) plans. Trace requirements to the model, design, and V&V to ensure all requirements were addressed. This is a complete design and would greatly assist your team in understanding and reviewing its completeness. Then proceed to the next level of abstraction in the requirements, model, design, and V&V. This process should be continued until all requirements are addressed.

Summary

The model-based approach to systems engineering offers clear advantages over the traditional paper-driven approach. The most obvious is that the database containing the model becomes the "corporate memory" for the project. Systems engineers change jobs often and take their project knowledge with them. With all rationale and decisions in the database, the knowledge stays with the project. Documents get lost and one is never sure whether the copy you have is the most up to date. This is not a problem with the model-driven approach.

The model also permits "what-if" analysis later. After the system is fielded, the customer might come to you and ask for the impact of making some upgrades to the system. With an up-to-date model, such an analysis is almost trivial.

Biography

Gerard H. Fisher is a Member of the Technical Staff at the Vitech Corporation, in Vienna, Va. He provides consulting services on the application of CORE® to support systems engineering, decision analysis, product development and program management. Prior to joining Vitech, Mr. Fisher was the corporate Director for Enterprise Process Improvement, for CACI. For 25 years, Mr. Fisher was a Senior Systems Engineer at IBM Federal Systems (and Loral after the acquisition) in Manassas, Virginia. He was involved in systems engineering on the Advanced Ballistic Missile Defense System and several high-performance signal processors for federal customers. He chairs the International Council on Systems Engineering (INCOSE) System Reengineering Working Group and is Technical Chair for the INCOSE WMA Chapter. He received his B.S. in Physics from St. Bonaventure University and M.S. in Systems Engineering from Virginia Tech.