SIEMENS

Ingenuity for life

Siemens PLM Software

# A holistic approach to vehicle system design

## Executive summary

This paper describes an approach to vehicle system design that uses standardized, hierarchical functions as a single level to describe electrical, electronic, and software content. Domain-specific implementation levels are then generated in a synthesis process, and evaluated using suitable metrics. The focus is on rapid, iterative optimization and on cross-domain architecture evaluation and validation

Hans-Juergen Mantsch, Siemens PLM Software

# Function-based system engineering

Functional approaches to describing and developing system architectures are often based on domain-specific languages derived from UML, such as EAST-ADL or SysML. At the same time, the technical content description (components of the system) appears in various forms and levels of abstraction (for example feature, activity, sequence, and/or status diagrams), and then suitably mapped for implementation.

This approach requires considerable effort and is less suited to architecture evaluation than to detailed documentation. Indeed, to be able to make meaningful technical and financial evaluations of the overall system architecture, each of the individual levels must be specified until a high degree of detail is achieved. In the subsequent mapping, the effort increases as the square of the level of detail: the number of artifacts within the individual levels, for example.

If the calculation of the corresponding metrics is not sufficiently agile, evaluation of a change in function allocation – for example of a software component on a particular control unit – cannot take place soon enough to provide truly meaningful results for each individual choice to be evaluated.

Overall, this significantly hampers architecture studies. The provision of the necessary data and calculations of the desired metrics can, in certain circumstances, take more time than planned for the entire project!

## Functional modeling

The alternative approach described here uses standardized, hierarchical function models combined on a single level to describe the technical content of system architecture. In this context, standardized means that individual functions can be separated from their eventual implementation as a hardware, driver, or software component. Instead of distributing the models across various, in some cases redundant, levels the individual domain-specific descriptions can be combined within a single functional abstraction, thereby eliminating the lengthy mapping process. Communication between individual functions is via signals that can be standardized as either software, electrical, or bus signals. All artifacts can be linked with a set of rules from a detailed options/variants model. The component models for hardware, software, and electrical and network communication can thereby be integrated, and their semantic dependencies checked and validated concurrently using design rules checks.

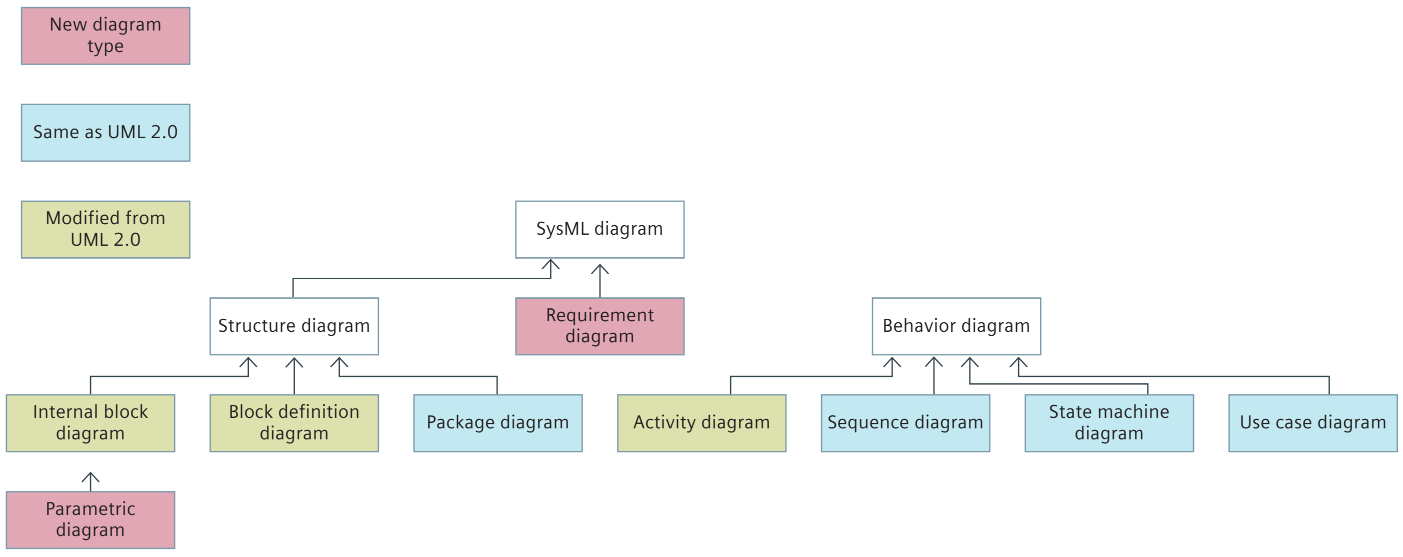


Figure 1: SysML diagram types (taxonomy), source Wikipedia.

In this way it is possible to capture the technical, variant-driven content of the downstream implementation domains (hardware, software, network, and electrical) as early as the functional abstraction level, and to validate this content across all variants.

To illustrate this approach, figure 3 shows a number of functional blocks. Software functions (SW), driver components (D), sensors (S), and actuators (A) are described and displayed within a single abstraction level. The signals between functions are shown according to their required implementation in the colors red (SW), green (electrical signals on a PCB), orange (electrical signals in a wiring harness), and blue (signals on a network).

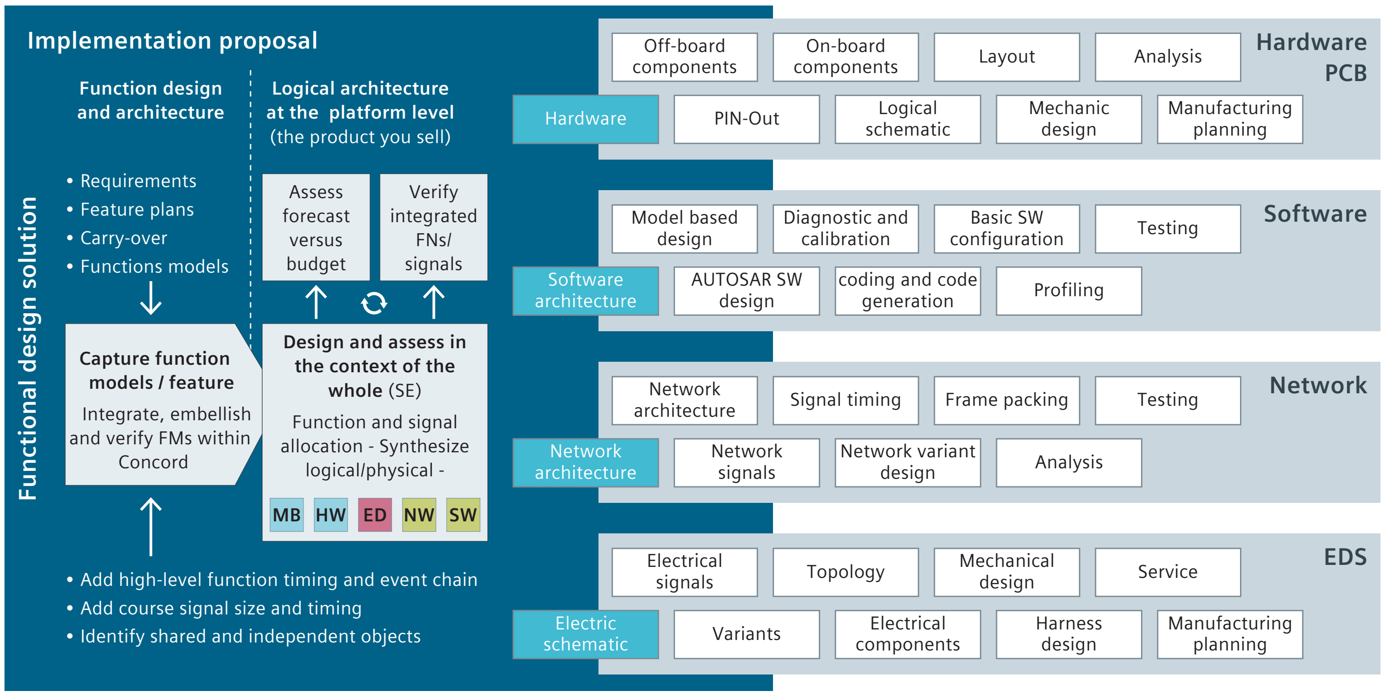


Figure 2: Domain-specific processes and upstream functional architecture design.

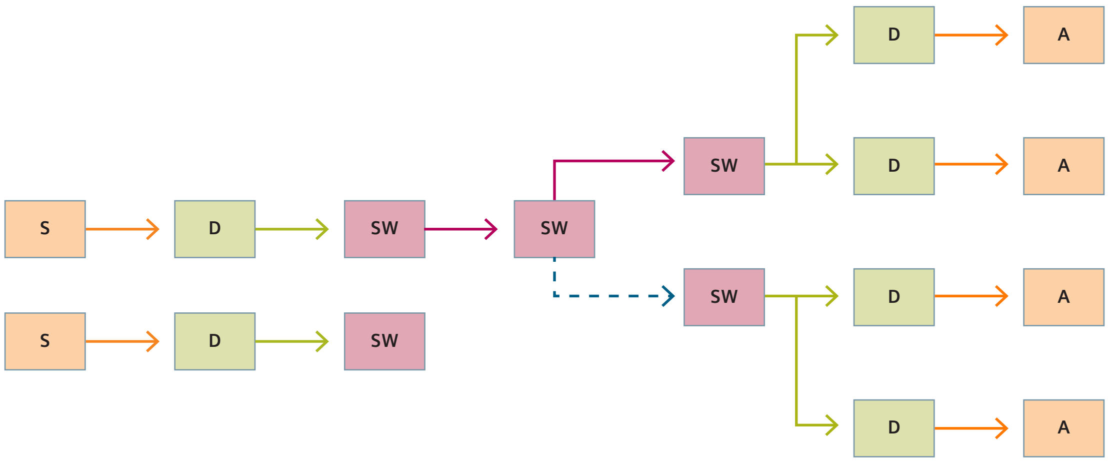


Figure 3: Functional design.

In figure 4 the individual type allocations correspond to implementation requirements for the downstream platform. If a function is of the software type, this means that the function is treated as a SW component in the downstream allocation to a platform: it should be allocated to a control unit and not to a purely electrical component. Note also that some of the functions and signals are optional, corresponding with the options/ variants model.

Functions can be organized hierarchically, and function signals can both reference their originating functions (if from an external functional design), and be made available across platforms and projects via a signal library.

## Logical platform

If the functional designs are captured as described, the downstream implementations (hardware and software, serial bus systems, and electrical distribution) can be created automatically, always respecting option/variant relationships.

To do so, first a logical platform is defined. This can be derived from a 3D model in the form of a physical topology, but can also start as an abstract logical network topology. Via the allocation of individual functional components to an options/variants model, a logical platform can encompass (in the example of automotive engineering) an individual car, a range of cars, or all possible derivatives of a car platform including the variation in software, electrical systems, network, and hardware. The same principle applies to trucks, offroad vehicles, aircraft and complex electromechanical machines such as industrial printers and medical equipment. Indeed, an extended system-of-systems such as an air defense system can be modeled in this way.

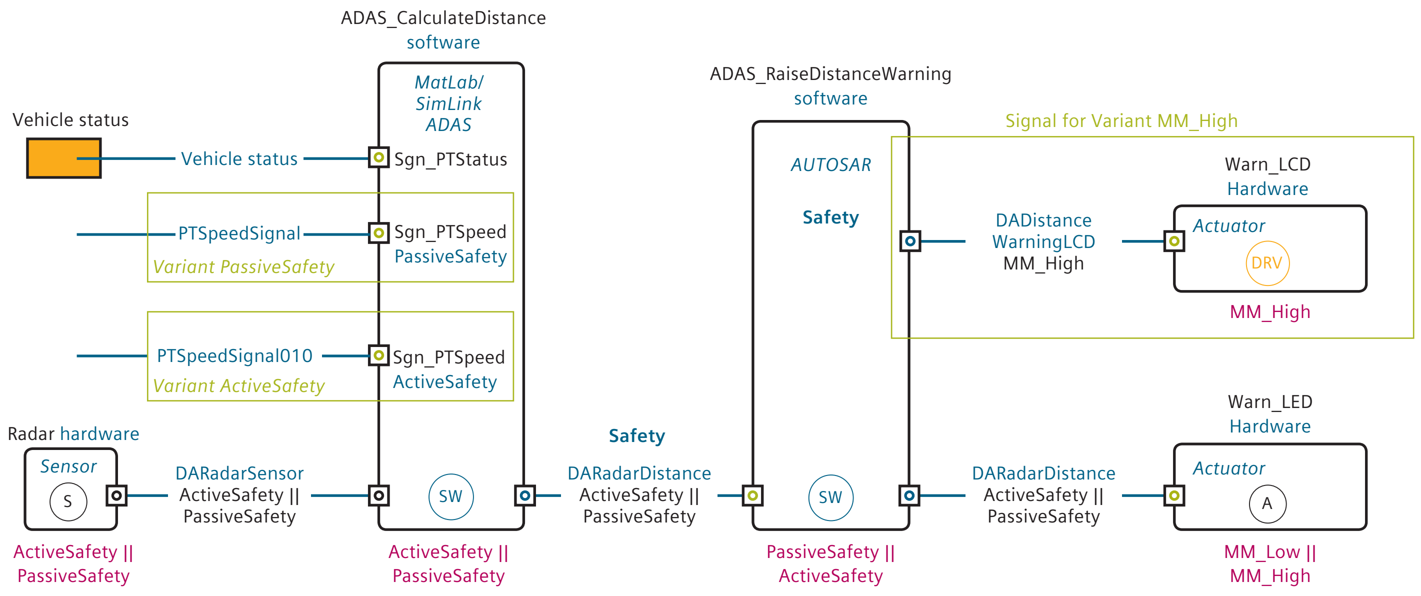

ADAS_CalculateDistance
software
MatLab/ SimLink ADAS
Vehicle status
Vehicle status
Sgn_PTStatus
PTSpeedSignal
Sgn_PTSpeed PassiveSafety
Variant PassiveSafety
PTSpeedSignal010
Sgn_PTSpeed ActiveSafety
Variant ActiveSafety
Radar hardware
Sensor
S
DARadarSensor ActiveSafety || PassiveSafety
SW
ActiveSafety || PassiveSafety
ActiveSafety || PassiveSafety
Safety
DARadarDistance ActiveSafety || PassiveSafety
ADAS_RaiseDistanceWarning
software
AUTOSAR
Safety
SW
PassiveSafety || ActiveSafety
Signal for Variant MM_High
DADistance WarningLCD MM_High
Warn_LCD
Hardware
Actuator
DRV
MM_High
DARadarDistance ActiveSafety || PassiveSafety
Warn_LED
Hardware
Actuator
A
MM_Low || MM_High

Figure 4: Function diagram with various functions, option allocations and references on external function blocks or signals.

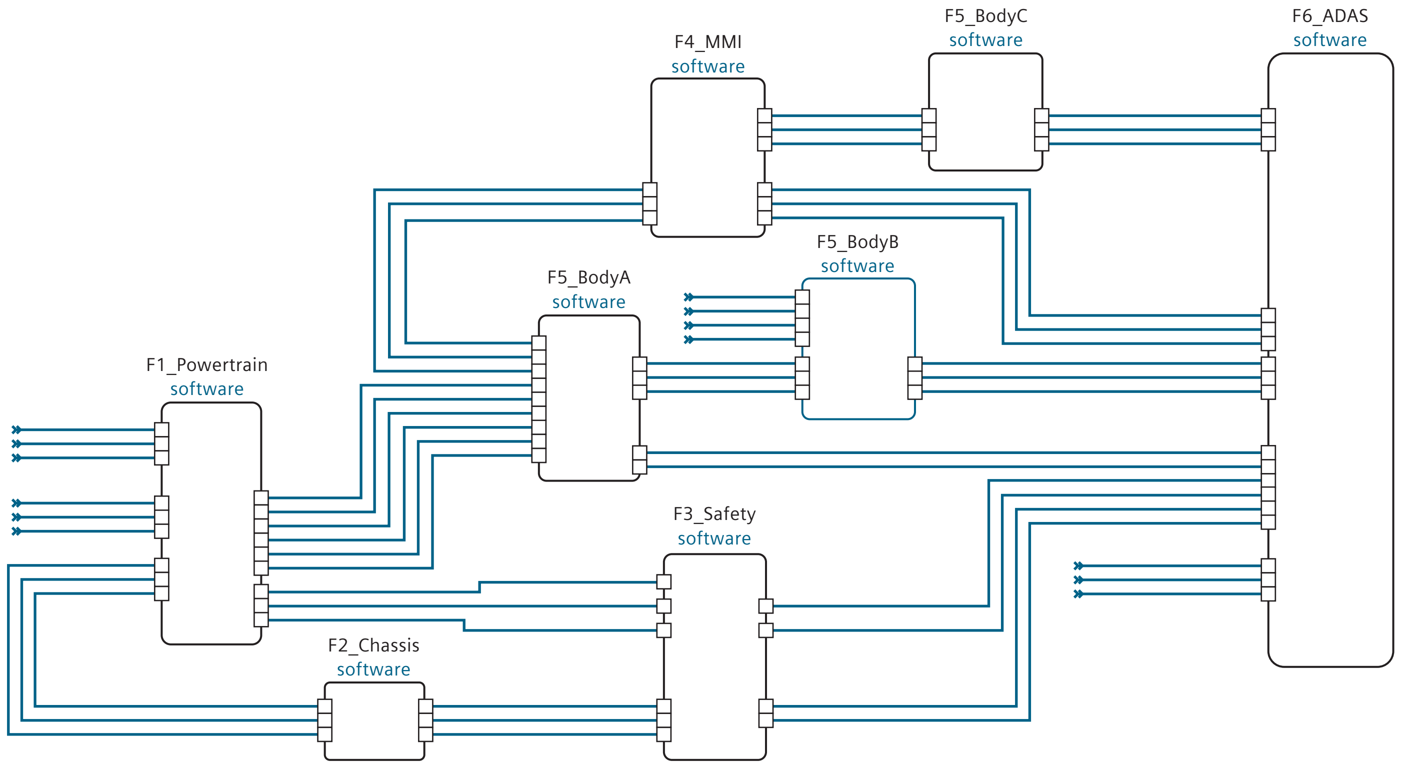


Figure 5: Diagram with various software-type functions.

The individual nodes of the platform are standardized as resources: electronic control units (ECUs) or line replaceable units (LRUs), electric assemblies, and electricity or earth conductors. They can be coupled electrically or via bus systems (CAN, LIN, Flexray, Ethernet, ARINC 429 etc), or indeed by optical or radio connections. These communication pathways are referred to as carriers.

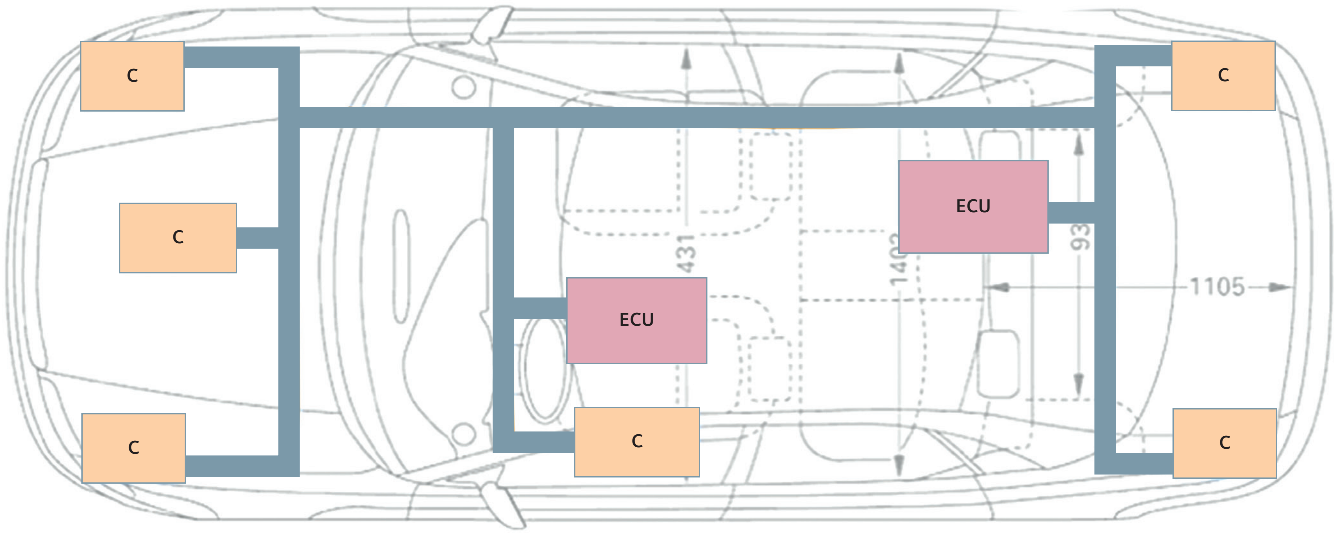
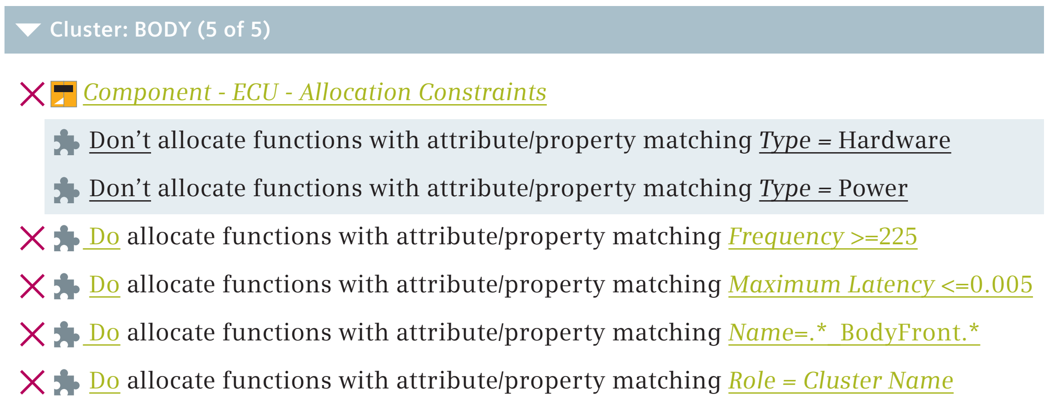


Figure 6: Platform architecture with the standardized function containers (resources) and connection pathways (carriers).

## Synthesis

Functions are then allocated into the logical platform. This can be done manually or automatically using rules. While doing so the functions are interrogated as to their type. For example, a software component is created from a function of the SW type, and then allocated to a control unit. The signals passing between functions are assigned to carriers as software, electrical, or network signals within the logical platform.

The resulting synthesis is the integrated implementation across the four domain types (hardware, software, network communication, and electrical) of the functional description. Semantic consistency can be analyzed in real time using design rule checks, and any necessary warnings or error messages generated.



Cluster: BODY (5 of 5)

*Component - ECU - Allocation Constraints*

Don't allocate functions with attribute/property matching *Type* = Hardware

Don't allocate functions with attribute/property matching *Type* = Power

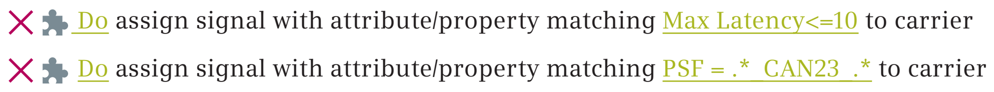
Do allocate functions with attribute/property matching *Frequency* >=225

Do allocate functions with attribute/property matching *Maximum Latency* <=0.005

Do allocate functions with attribute/property matching *Name*=.*_BodyFront.*

Do allocate functions with attribute/property matching *Role* = *Cluster Name*

Figure 7: Rules for allocating functions.



Do assign signal with attribute/property matching Max Latency<=10 to carrier

Do assign signal with attribute/property matching PSF = .*_CAN23_.* to carrier

Figure 8: Rules for allocating functions and signals.

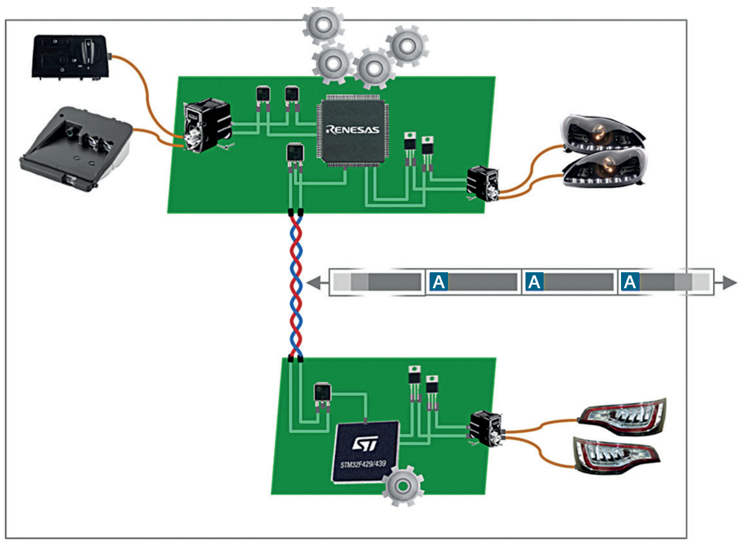


Figure 9: Sample diagram of synthesis implementation.

# Metrics

As early as this synthesis process, metrics for technical evaluation can be calculated. These metrics can be configured to show a wide variety of information. For example, for multiplex networks interesting metrics include load, tolerance, and overhead. For the electrical domain they include the number of wires, splices and connectors, wire lengths, and bundle diameter. For control units they include device weight, CPU load, requirements for RAM, ROM, FLASH/EEPROM, PCB area and volume power, and thermal dissipation. The metrics are calculated from parameters attached to functions, resources, and carriers: these parameters will often be well known from previous implementations.

If a value is above a particular level, for example if the forecast requirement for RAM goes beyond the budget provided by the microprocessor, alerts will be issued via the design rule checks or directly into the platform architect's graphical display. This helps the engineer ensure the design is feasible.

Furthermore, not just technical metrics can be calculated. By extending the calculations project goals such as cost, weight, headroom, reliability or re-use can also be reckoned.

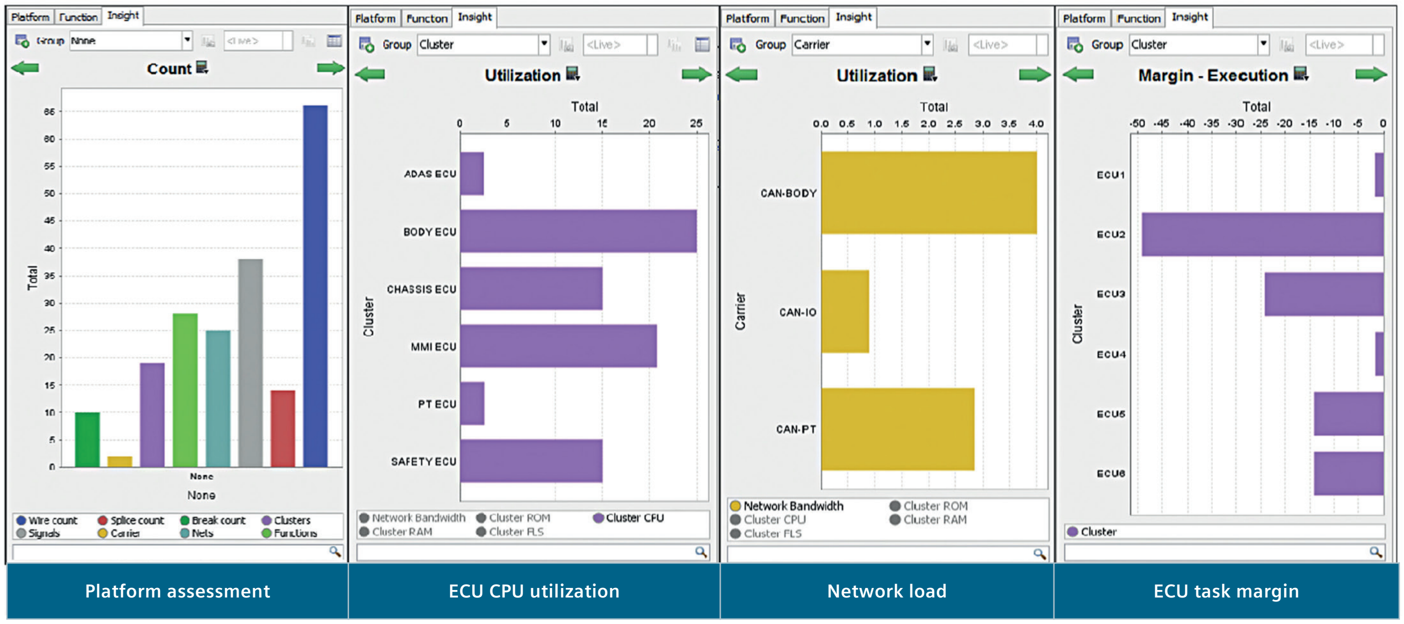


Figure 10: Example metrics: object count, CPU utilization, network load, and task scheduling.

# Evaluation and optimization

Because evaluation using these metrics is done in real time, ie when design decisions or changes are made, this process is ideally suited to evaluating alternative implementations (“architectures”), or indeed revised functional content. Changes are immediately reflected into the metrics, and alternative strategies can then be studied. Issues of optimum functional partitioning, electrical optimization, cost, and runtime optimization can thus be addressed iteratively and interactively.

After the final evaluation, the results of the logical platform synthesis are fed into the downstream, detailed design process in each domain-specific format such as ARXML, FIBEX, or KBL. The results of the architecture study phase can thus be re-used as implementation suggestions for future platforms. In the case of an integrated design environment, data can of course be passed directly to the appropriate applications.

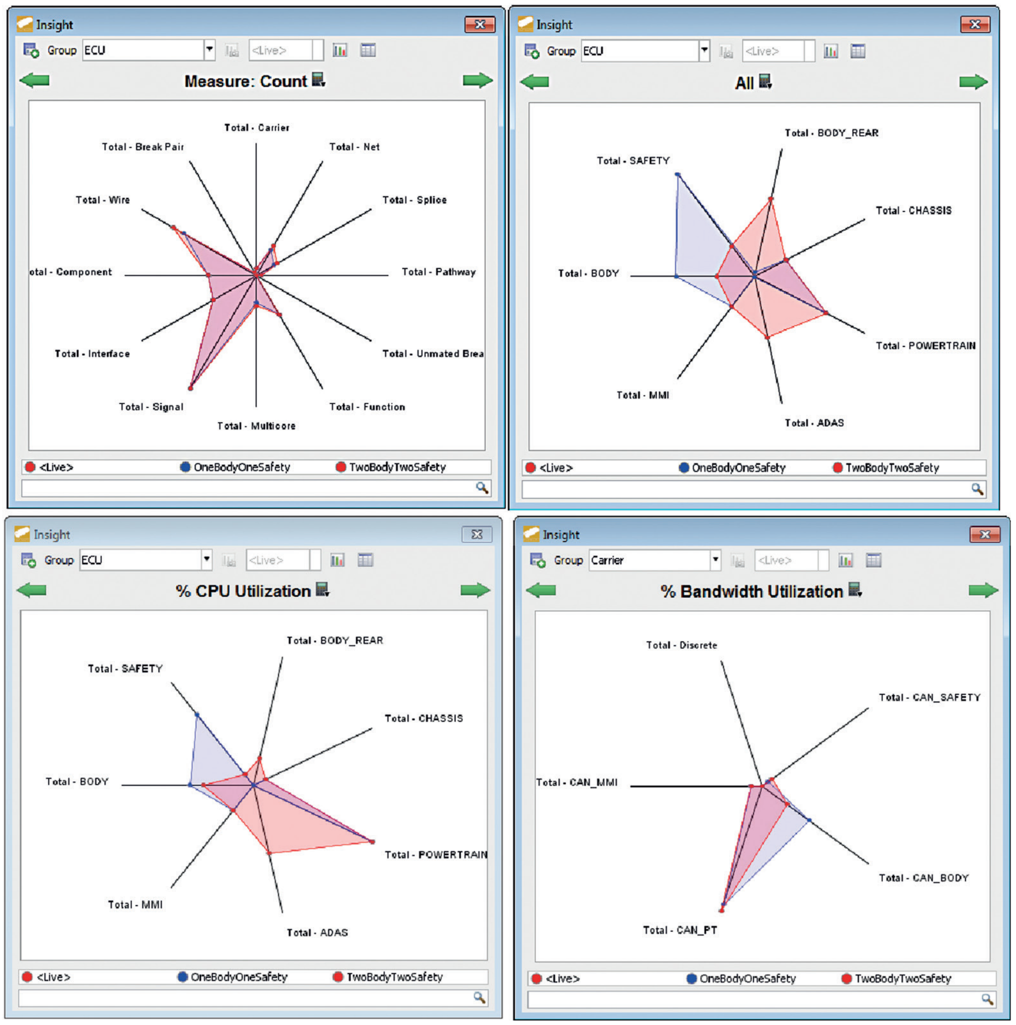


Figure 11: Comparison of different expansion and optimization stages in terms of the object count, technical evaluation, CPU, and network traffic measurements.

# Tying it together

The approach described in this paper uses the functional abstraction to consolidate the various E/E domains at a single level. This in turn allows rapid evaluation of implementation alternatives, while preparing data for use in detailed design.

For such architecture evaluation and validation, existing approaches based on UML or SysML-like meta-models are less suitable, because of the technical effort and knowledge needed. The related complexity allows virtually no scope for achieving the adequate or necessary level of detail for a comprehensive evaluation in the available time.

Commercial software based on the principles described in this paper is available within Siemens PLM Software product suite.

By contrast, the approach described uses a functional abstraction in which the implementation-related data and artifacts are combined into standardized, functional models, instead of distributing them across different, in some cases redundant levels.

As early as during the automated allocation to logical platforms, the models can be iteratively validated for feasibility of implementation and safeguarded with corresponding technical and commercial metrics. The result of the architecture process is implementation suggestions for the downstream development processes for software, network, electrical systems, and hardware.

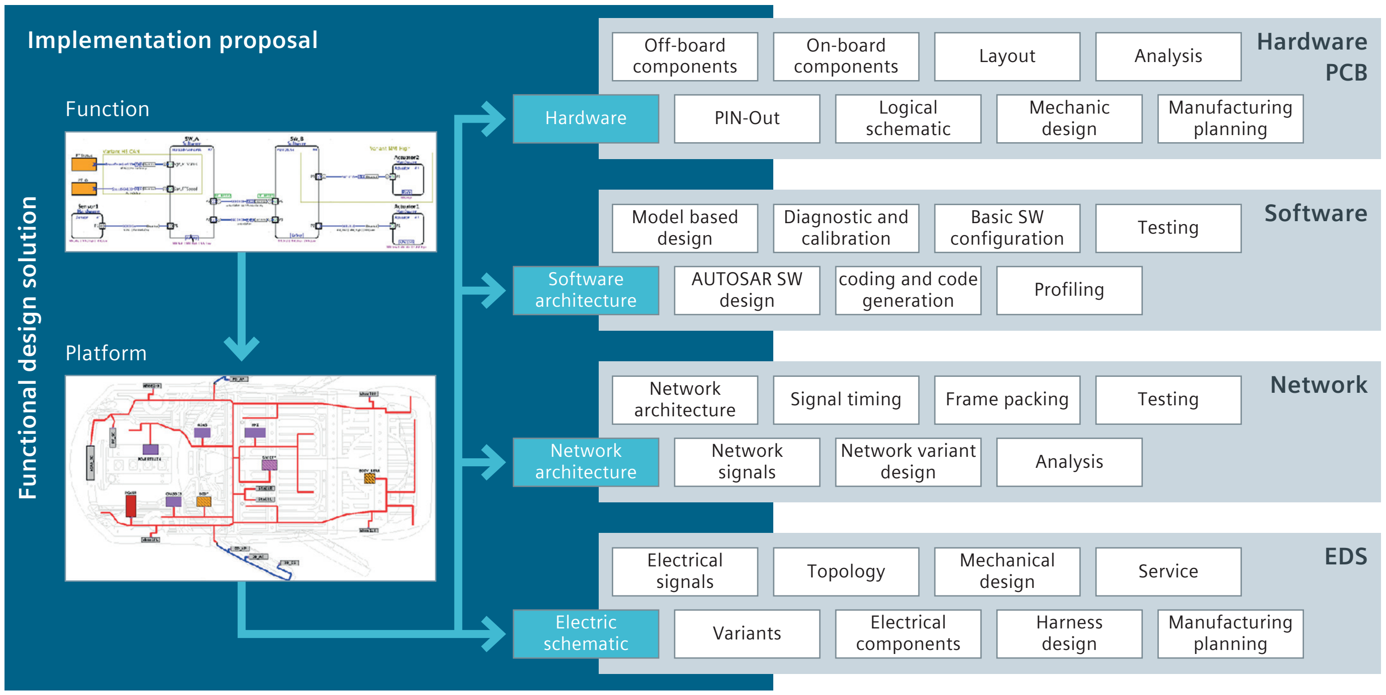


Figure 12: Functional architecture design and assessment, and resulting implementation proposals for downstream design flow.

**Siemens PLM Software**

**Headquarters**
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

**Americas**
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

**Europe**
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

**Asia-Pacific**
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

## About Siemens PLM Software

Siemens PLM Software, a business unit of the Siemens Digital Factory Division, is a leading global provider of software solutions to drive the digital transformation of industry, creating new opportunities for manufacturers to realize innovation. With headquarters in Plano, Texas, and over 140,000 customers worldwide, Siemens PLM Software works with companies of all sizes to transform the way ideas come to life, the way products are realized, and the way products and assets in operation are used and understood. For more information on Siemens PLM Software products and services, visit siemens.com/plm.

**siemens.com/plm**

77040-A4 3/19 Y